6. EXAMPLE PROGRAMS

Included with the M-API software are some example programs. These programs serve two purposes:

1. They offer specific examples of how to use the M-API routines.
2. They give the user an opportunity to verify, after installation, that the M-API routines were built correctly and are functioning as expected.

The example programs can be built by typing: "make test" while in the examples directory. The example programs can be executed, and if necessary built, with the output piped to the file "example.output" by typing: "make run". Individual routines can be built by typing "make *name*". The user will have to edit the makefile to work as expected for their version on UNIX.

NOTE: *Name* must be replaced with the example program name (e. g., example1).

The following example programs create, write, and read MODIS HDF arrays and tables.

6.1 Example 1: Creating a Floating Point Array in FORTRAN

This FORTRAN program demonstrates how to create a 32-bit floating point array. The HDF file " arrex1.hdf" is created using OPMFIL. A MODIS group, "Yale_class", is created using CRMGRP. The last dimension of the array is given the name "Wade_dimension". An initialized array containing 1's is created with CRMAR and then written to the HDF file with PMAR under the previously created MODIS group. Once the file has been written it is closed with a call to CPMFIL. CPMFIL is used since it is a new HDF file.

List of routines called:

Name	Description
OPMFIL	Opens a MODIS file (file access r, w, a).
CRMGRP	Creates a Vgroup.
CRMAR	Creates an array.
PMDNAM	Adds a dimension name to the last dimension of the array.
PMAR	Writes the array to a file.
CPMFIL	Completes and closes a MODIS file.

6.1.1 Source Code Listing for Example 1

```
      PROGRAM example1
      IMPLICIT NONE
      INCLUDE 'mapi.inc'

C This example program demonstrates how to open a new MODIS HDF file,
C create a data array in a Vgroup, write data to the array, put a
C dimension name to the last dimension of it, and close the MODIS HDF
C file.
C

C  DATA ARRAY
      REAL DATF(32,64)

C  MODIS FILE POINTER ARRAY
      INTEGER MODFIL(MODFILLEN)

C  DIMENSION ARRAY
      INTEGER DIMS(2)

C  Start indices (0-based) for writing array
      INTEGER STA(0:1)

C  rank and error code
      INTEGER RANK, IER

C  dimension to be named (0-based)
      INTEGER DIM

C  dimension name to be written
      CHARACTER*(100) DNM

C  Array and group names
      CHARACTER*20 FILNM, ARRNM, GRPNM, CLASSNAME

C  Data type
      CHARACTER*(DATATYPELENMAX) DTYPE

C  MDHANDLES array of ECS metadata groups in MCF
      CHARACTER*(PGSd_MET_GROUP_NAME_L - 1)
     +             MDHANDLES(PGSd_MET_NUM_OF_GROUPS)

C  names of local attributes to store ECS metadata in
      CHARACTER*(MAX_ECS_NAME_L - 1)
     +             HDFATTNMS(PGSd_MET_NUM_OF_GROUPS)

C  Number of handles
      INTEGER NUMHANDLES

C  Initialize values
      DATA DATF /2048*1/
      DATA STA /2*0/
```

```
      DATA DIMS /32,64/
      DATA RANK /2/
      DATA ARRNM /'DATAFLOAT'/
      DATA GRPNM /'Qi_group'/
      DATA CLASSNAME /'Yale_class'/
      DATA FILNM /'arrex1.hdf'/
      DATA NUMHANDLES /0/
      DATA DIM /0/
      DATA DNM /'Wade_dimension'/
      DTYPE = R32

      print*,'*** Example1 ***'

C  Open file
      IER = OPMFIL(FILNM, CREATE_FILE, MODFIL)

      IF(IER.EQ.MAPIOK) THEN
           PRINT *,'Openning of Modis file is successful!'
      END IF

C  Create a Vgroup
      IER = CRMGRP(MODFIL,GRPNM,CLASSNAME)

      IF(IER.EQ.MAPIOK) THEN
          PRINT *,'Creating Vgroup is successful!'
      END IF

C  Create array
      IER = CRMAR(MODFIL, ARRNM, GRPNM, DTYPE, RANK, DIMS)

      IF(IER.EQ.MAPIOK) THEN
          PRINT *,'Creating an array is successful!'
      END IF

C  Put dimension name to the last dimension of the array
      IER = PMDNAM(MODFIL,ARRNM,GRPNM,DIM,DNM)

      IF(IER.EQ.MAPIOK) THEN
           PRINT *,'Putting a dimension name is successful!'
      END IF

C  Write to the array (note that the entire array is being written, so
C  data dimensions are equal to array dimensions)
      IF(IER .EQ. MAPIOK) THEN
           PRINT *,'Writing array to MODIS HDF file!'
      IER = PMAR(MODFIL,ARRNM,GRPNM,STA,DIMS,DATF)
      END IF
C     PRINT *,'Wrote array to MODIS HDF file!'

C New MODIS file so use CPMFIL to close the file
      IER = CPMFIL(MODFIL, MDHANDLES, HDFATTNMS, NUMHANDLES)

      IF(IER .EQ. MAPIOK) THEN
           PRINT *,'MODIS file was successfully closed!'
```

```
      END IF
      PRINT *,'example1 done'
      PRINT *,' '
      STOP
      END

C  End of example
```

6.2 Example 2: Creating a Floating Point Array in C

This C program is similar to the previous FORTRAN program. This program demonstrates how to create a 32-bit floating point array. When the program executes, a HDF file "arrex2.hdf" is opened. A MODIS group, "Qi_group" is created. The last dimension of the array is named "Wade_dimension". A 64 by 32 element array is created and then initialized to floating point values ranging from 0 to 2048. The array is then written to the HDF file under the previously created MODIS group. Once the file has been written the file is completed since it is a new HDF file.

List of routines called:

Name	Description
openMODISfile	Opens a MODIS file (file access r, w, a).
createMODISgroup	Creates a Vgroup.
createMODISarray	Creates an array.
putMODISdimname	Adds a dimension name to the last dimension of the array.
putMODISarray	Writes the array to a file.
completeMODISfile	Completes and closes a MODIS file.

6.2.1 Source Code Listing for Example 2

```
#include <stdlib.h>
#include <stdio.h>
#include "mapi.h"

/*    This example program demonstrates how to open a new MODIS HDF file,
      create a data array in a Vgroup, write data to the array, put a
      dimension name to the first dimension of it, and close the MODIS HDF
      file. */

main()
   {
   MODFILE  *modfile;         /* Modis file pointer */
   float32 data[64][32];               /* Data Array */
   long cksum = 0;                     /* array check sum  */
   long dims[2] = {64,32};          /* Array dimensions */
   long sta[2]  = {0,0};            /* Array start indices (0-based) */
   long rank    = 2;                /* Array rank */
   char dtype[] = R32;                 /* Array type */
   char arrnm[] = "DFLOAT";         /* Array name */
   char grpnm[] = "Qi_group";       /* Group name */
   char fname[] = "arrex2.hdf";        /* Modis file name */
   char faccess[] = "w";               /* Modis file access */
```

```
int mapier;                          /* Error code */
int i,j;                   /* counters */
long dimension = 0;
char dimname[] = "Wade_dimension";

PGSt_MET_all_handles     mdHandles;
ECSattr_names_for_all_handles HDFattrnms;
long NumHandles = 0;

printf(" *** Example2 ***\n");

/* open the file */
modfile= openMODISfile(fname, faccess);
if (modfile==NULL)
{
   printf ("Error openning %s, exiting \n",fname);
   exit(-1);
}
else
  printf(" File: %s opened, access mode %s\n",fname,faccess);

mapier = createMODISgroup(modfile, grpnm, NULL);
if (mapier == MFAIL)
{
   printf ("Error creating group, exiting\n");
   exit(-1);
}
else
  printf(" Group created, Name: %s,\n", grpnm);

mapier = createMODISarray(modfile,arrnm,grpnm,dtype,rank,dims);
if (mapier == MFAIL)
{
   printf ("Error creating array, exiting\n");
   exit(-1);
}
else
  printf(" Array created, Name: %s,\n",arrnm);

/* Put dimnesion name to the array */
mapier = putMODISdimname(modfile,arrnm,grpnm,dimension,dimname);
if ( mapier == MFAIL )
{
  printf("Error writing dimension name, exiting\n");
  exit(-1);
}
else
  printf("putMODISdimname is successful\n");

/* Write to the array (note that the entire array is being
   written, so data dimensions are equal to array dimensions */

for (i=0; i< dims[0]; i++)
  for (j=0; j< dims[1]; j++)
```

```
     {
       data[i][j]= (float32)((i+j) + 1000.0);
       cksum = cksum + data[i][j];
     }

   mapier = putMODISarray(modfile,arrnm,grpnm,sta,dims,data);
   if (mapier == MFAIL)
   {
     printf ("Error writing array, exiting\n");
     exit(-1);
   }
   else
   {
     printf(" Array check sum: %d \n",cksum);
     printf(" Put the array in the file...\n");
   }

   /* Close the MODIS-HDF file */
   mapier = completeMODISfile(&modfile, mdHandles, HDFattrnms, NumHandles);
   if (mapier == MFAIL){
      printf ("Error closing file, exiting\n");
       exit(-1);
   }else{
     printf(" File closed successfully\n");
   }
   printf("\n");
   exit(0);
}
/* End of example */
```

6.3 Example 2a: Open an HDF File to Read and Print an Array

This C program reads the array that example2 wrote to disk. The MODIS file is opened. The array is retrieved and printed. The file is then closed. When reading an array it is necessary to pass specific information about the array (i.e., rank, dimensions, and datatype) to the routine. If this information is not known, then getMODISardims can be used to retrieve the required array information.

List of routines called:

Name	Description
`openMODISfile`	Opens a MODIS file (file access: r, w, a).
`getMODISarray`	Retrieves an array or subarray.
`closeMODISfile`	Closes a MODIS file.

6.3.1 Source Code Listing for Example 2a

```
#include <stdlib.h>
#include <stdio.h>
#include "mapi.h"

/*
**  This example program  demonstrates how to open a MODIS HDF file,
**  read a given data array from the file, and close the file.
*/

main()
{
   MODFILE  *modfile;         /* Modis file pointer */
   float data[64][32];              /* Data Array */
   long dims[2] = {64,32};          /* Array dimensions */
   long sta[2] = {0,0};       /* Array start indices (0-based) */
   long rank = 2;             /* Array rank */
   char dtype[] = "float32";           /* Array type */
   char arrnm[] = "DFLOAT";         /* Array name */
   char grpnm[] = "Qi_group";            /* Group name */
   long ier;                          /* Error code */
   int i,j;                   /* counters */
   int cksum = 0;

   printf(" *** Example2a ***\n");

   /* Open the MODIS-HDF file */
   modfile= openMODISfile("arrex2.hdf","r");
   if (modfile==NULL) exit(1);
   else
     printf("openMODISfile is successful\n");
```

```
   ier = getMODISarray(modfile,arrnm,grpnm,sta,dims,data);
   printf("ier(getMODISarray) = %d\n",ier);
   if (ier == -1) exit(1);

   for (i=0; i<64; i++)
     for (j=0; j<32; j++)
       cksum += data[i][j];
   printf("cksum = %d\n",cksum);

   /* Close the MODIS-HDF file */
   ier = closeMODISfile(&modfile);
   printf("ier(closeMODISfile) = %d\n",ier);
   if (ier == -1) exit(1);

   printf("\n");
   exit(0);
   }
/* End of example */
```

6.4 Example 3: Create an Integer Array in FORTRAN

This FORTRAN program demonstrates how to create a 32-bit integer array. The HDF file "arrex3.hdf" is created using OPMFIL. A MODIS group, "Yale_class", is created using CRMGRP. The last dimension of the array is given the name "RECORD_NUMBER". An initialized array containing 1's is created with CRMAR and then written to the HDF file with PMAR under the previously created MODIS group. Once the array has been written the file is closed with a call to CPMFIL. CPMFIL is used since it is a new HDF file.

List of routines called:

Name	Description
OPMFIL	Opens a MODIS file (file access r, w, a).
CRMGRP	Creates a Vgroup.
CRMAR	Creates an array.
PMDMIN	Writes an array dimension name to a file.
PMAR	Writes the array to a file.
CPMFIL	Completes and closes a MODIS file.

6.4.1 Source Code Listing for Example 3

```
C EXAMPLE 3:  Create a 32-bit integer array in a Vgroup and write data
C to it, also put an attribute to the last dimension of this array

            PROGRAM example3
            IMPLICIT none
            INCLUDE 'mapi.inc'

C  DATA ARRAY
      INTEGER IDATA(15,20)
C  Counter
      INTEGER I
C  MODIS FILE POINTER ARRAY
      INTEGER MODFIL(MODFILLEN)
C  DIMENSION ARRAY
      INTEGER DIMS(3)
C  Start indices (0-based) for writing array
      INTEGER STA(3)
C  rank and error code
      INTEGER RANK, IER
C  Number of handles
            INTEGER NUMHANDLES
```

```
C  Array and group names
      CHARACTER*20 ARRNM, GRPNM, FILNM, CLASSNAME
C  Dimension name
      CHARACTER*20 DIMNM
C  Data type
      CHARACTER*(DATATYPELENMAX) DTYPE, ATYPE
C  Array Data type, array attribute
      CHARACTER*20 ATTR
C  Attribute value
      CHARACTER*100 ATTRV
C  mdHandles array of ECS metadata groups in MCF
            CHARACTER*(PGSd_MET_GROUP_NAME_L - 1)
     +             mdHandles(PGSd_MET_NUM_OF_GROUPS)
C  names of local attributes to store ECS metadata in
            CHARACTER*(MAX_ECS_NAME_L - 1)
     +             hdfattnms(PGSd_MET_NUM_OF_GROUPS)

      DATA IDATA/300*1/
      DATA DIMS/15,20,100/
      DATA STA/3*0/
      DATA RANK/3/
      DATA ARRNM /'DATASHORT'/
      DATA GRPNM /'Qi_group'/
      DATA CLASSNAME /'Yale_class'/
      DATA FILNM /'arrex3.hdf'/
      DATA DIMNM /'RECORD NUMBER'/
      DATA ATTR /MLONG_NAME/
      DATA ATTRV /'This is the attribute value'/
      DATA ATYPE /'CHARACTER*(*)'/
      DATA NUMHANDLES /0/
      DATA DTYPE /I32/

      print*,"*** Example3 ***"
C     Open file
      IER = OPMFIL(FILNM, CREATE_FILE, MODFIL)

      IF(IER.EQ.MAPIOK) THEN
         PRINT *,'Openning of Modis file is successful!'
      END IF

C     Create a Vgroup
      IER = CRMGRP(MODFIL,GRPNM,CLASSNAME)
      IF(IER.EQ.MAPIOK) THEN
         PRINT *,'Creating Vgroup is successful!'
      END IF

C  Create the array
      IER = CRMAR(MODFIL,ARRNM,GRPNM,DTYPE,RANK,DIMS)
      PRINT *,'IER(CRMAR) = ',IER

C     Put an attribute to the last dimension
      IER = PMDMIN(MODFIL,ARRNM,GRPNM,0,ATTR,ATYPE,43,ATTRV)
      PRINT *,'IER(PMDMIN) = ',IER
```

```
      IF (IER .EQ. MAPIOK) THEN
          PRINT *,'Writing array to MODIS HDF file!'
C     Re-define the last dimension for writing the array
          DIMS(3) = 1
C     Loop on the last dimension to write the array.
          DO I=1,100
C     and write to the array
             STA(3) = I - 1
             IER = PMAR(MODFIL,ARRNM,GRPNM,STA,DIMS,IDATA)
          END DO
       ENDIF
C      Close HDF file
       IER = CPMFIL(MODFIL, MDHANDLES, HDFATTNMS, NUMHANDLES)
c      IER = CLMFIL(MODFIL)
       IF(IER .EQ. MAPIOK) THEN
          PRINT *,'MODIS file was successfully closed!'
       END IF

       PRINT *,' '
C      End of example

       STOP
       END
```

6.5 Example 4: Create an Integer Array in C

This C program is similar to the previous FORTRAN program. This program demonstrates how to create a 16-bit integer point array. When the program executes, a HDF file "arrex4.hdf" is created. A MODIS group, "Qi_group", is also created. An attribute "attr value" is attached to the last dimension of the array. A 20 by 15 element array is created with CRMAR and then initialized to the integer value 1. The array is then written to the HDF file under the previously created MODIS group. Once the array has been written the file is closed.

List of routines called:

Name	Description
openMODISfile	Opens a MODIS file (file access r, w, a).
createMODISgroup	Creates a Vgroup.
createMODISarray	Initializes a MODIS HDF array.
putMODISdiminfo	Attaches a local attribute /value pair to a specific dimension of a MODIS array.
putMODISarray	Writes an array or subarray to a MODIS file.
closeMODISfile	Closes a MODIS file.

6.5.1 Source Code Listing for Example 4

```
#include <stdio.h>
#include "mapi.h"

/*   This example program demonstrates opening a new MODIS HDF file,
     creating a data array in a Vgroup, writing data to the array, putting
     an attribute to the first dimension of it, and closing the HDF file. */

main()
{
   MODFILE      *modfile;           /* Modis file pointer */
   char filename[] = "arrex4.hdf";  /* HDF file name */
   int16  idata[20][15];              /* Data Array */
   long dims[3] = {100,20,15};      /* Array dimensions */
   long sta[3] = {0, 0, 0};              /* Array start indices (0-based) */
   long rank = 3;             /* Array rank */
   char dtype[] = "int16";          /* Array type */
   char arrnm[] = "DATASHORT";           /* Array name */
   char attrname[]  = "long_name";      /* Array attribute */
   char attrvalue[] = "attr value";     /* Array attribute value */
   char attr_dtype[] = "char *";             /* Array attribute data type */
```

```
   char grpnm[] = "Qi_group";           /* Group name */
   char classname[] = "Yale_class";
   long ier;                            /* Error code */
   int i,j,k;                           /* counters */

   /* Set all idata's value to 1 */
   for (i=0; i<20; i++)
     for (j=0; j<15; j++)
       idata[i][j] = (int16)1;
   printf(" *** Example4 ***\n");

   /* Create the MODIS-HDF file */
   modfile= openMODISfile(filename, "w");
   if (modfile==NULL) exit(1);

   /* Create a Vgroup */
   ier = createMODISgroup(modfile,grpnm,classname);
   printf("ier(createMODISgroup) = %d\n",ier);

   /* Create array */
   ier = createMODISarray(modfile,arrnm,grpnm,dtype,rank,dims);
   printf("ier(createMODISarray) = %d\n",ier);
   if (ier == -1) exit(1);

   /* Put attribute info to the first dimension */
   ier = putMODISdiminfo(modfile,arrnm,grpnm,0,attrname,attr_dtype,
                 strlen(attrvalue),attrvalue);
   printf("ier(putMODISdiminfo) = %d\n",ier);
   if (ier == -1)
     exit(1);

   /*  Re-define the last dimension for writing the array */
   dims[0] = 1;

   /*  Loop on the last dimension to write the array. */
   for (i=0; i<=99; i++)
   {
      sta[0] = i;
      /*  Set the start index for the last dimension and write to
          the array                  */
      ier = putMODISarray(modfile,arrnm,grpnm,sta,dims,idata);
      if (ier == -1) exit(1);
   }

   /* Close the MODIS-HDF file */
   ier = closeMODISfile(&modfile);
   printf("ier(closeMODISfile) = %d\n",ier);

   printf("\n");
   exit(0);
}
```

6.6 Example 5: Read an Integer Array

This FORTRAN program demonstrates how to read a 32-bit integer array. The HDF file "arrex3.hdf" (created by example3) is opened for reading using OPMFIL. The array was stored under the MODIS group QI_group, so this must be specified when retrieving the array. The array is read in by looping on the second dimension of the array using GMAR. Once the array has been read into memory, the file is closed with a call to CLMFIL. It should also be noted that even though a C program was used to write this array to disk, a FORTRAN program can be used to retrieve it.

List of routines called:

Name	Description
OPMFIL	Opens a MODIS file (file access r, w, a).
GMAR	Retrieves an array from an HDF file.
CLMFIL	Closes preexisting MODIS file.

6.6.1 Source Code Listing for Example 5

```
      program example5
c EXAMPLE 5:  Read the array from the previous example by
c looping on the second array index, using FORTRAN.

      INCLUDE 'mapi.inc'
c  DATA ARRAY
      INTEGER JDATA(15,100)
C Array Checksum
      INTEGER cksum
c  MODIS FILE POINTER ARRAY
      INTEGER MODFIL(MODFILLEN)
C  DIMENSION ARRAY
      INTEGER DIMS(3)
C  Start indices (0-based) for reading array
      INTEGER STA(3)
C  Error code
      INTEGER IER
C Array and group names
      CHARACTER*20 ARRNM, GRPNM, FILNM
      DATA DIMS/15,20,100/
      DATA STA/3*0/,cksum/0/
      DATA ARRNM/'DATASHORT'/
            DATA GRPNM/'Qi_group'/
      DATA FILNM/'arrex3.hdf'/

      print*,'*** Example5 ***'
```

```
C       Open file
      IER = OPMFIL(FILNM, 'r', MODFIL)
      IF (IER .EQ. MAPIOK) THEN
C     Re-define the second dimension for reading the array
         DIMS(2) = 1
         i=1
C     Loop on the second dimension to read the array.
         DO while (i .le. 20 .and. ier .eq. MAPIOK)
C     Set the start index for the second dimension and read
C     the array

      STA(2) = I - 1
      IER = GMAR(MODFIL,ARRNM,GRPNM,STA,DIMS,JDATA)
         i = i+1
      END DO
      IF (IER .EQ. MAPIOK) THEN
         do l=1,15
         do m=1,100
                cksum = cksum + jdata(l,m)
         end do
         end do
            print*,'Array retrieved: ',arrnm
            print*,'Checksum: ',cksum
         else
            print*,'GMAR: failed @ I=',i
      ENDIF
      ENDIF

C     Close file
      IER = CLMFIL(MODFIL)
      IF (ier .ne. MFAIL)THEN
         print*,'File closed.'
      ELSE
         print*,'Error closing file.'
      ENDIF

      print *,' '
C     End of example
      STOP
      END
```

Table 6-1 structure is created with the name "Bolide Heights". The data group argument is set to NULL so the data structure is not placed in any MODIS group.

Table 6-1. Sample Data Table
Bolide Heights

Record Number	Latitude (degrees)	Longitude (degrees)	Altitude (m)
Number Type	float32	float32	int32
0	40.2	-77.8	23500
1	-22.8	132.5	37000
2	63.2	93.6	2200

The following example routines create, read, and write MODIS HDF tables.

6.7 Example 6: Create a MODIS HDF Table

This FORTRAN program demonstrates how to create a MODIS HDF table. The table consists of three columns and three rows (see Table 6-1 Sample Data Table). Two columns are real data and one column is integer data. As in the previous examples an HDF file ("tblex6.hdf") is openned for writing using OPMFIL. An HDF table is created using CRMTBL. The table is then written to the HDF file using PMTBL. Once the file has been written the file is closed with a call to CLMFIL.

List of routines called:

Name	Description
`OPMFIL`	Opens a MODIS file (file access r, w, a).
`CRMGRP`	Creates a Vgroup.
`CRMTBL`	Creates a table for accessing.
`PMTBL`	Writes a table to an HDF file.
`CPMFIL`	Completes a new MODIS file.

6.7.1 Source Code Listing for Example 6

```
C     This program will create a modis HDF table called "Bolide Heights"
C     in a Vgroup by using CMTBL, then write 3 records to the table by
C     using PMTBL.
C=======================================================
      program example6
      IMPLICIT NONE
      INCLUDE 'mapi.inc'

C DATA BUFFER
      byte             data1(12)
C MODIS file pointer array
      integer          mfile(MODFILLEN)
C Number of records to access and location of first record to access
      integer            recno, start
C Error code
      integer            ier
C File, table name, table class, and group names
      character*80     filen,tbname,group,classname
C Table field names
      character*80     field
C Data type, using M-API parameter to size string
      character*(3*DATATYPELENMAX) dtype
C Data arrays and type-matched buffers
      real             lat(3), lon(3), f1, f2
      integer          height(3), i3
      integer          i
C  mdhandles array of ECS metadata groups in MCF
      character*(PGSd_MET_GROUP_NAME_L - 1)
     +             mdhandles(PGSd_MET_NUM_OF_GROUPS)

C  names of local attributes to store ECS metadata in
      character*(MAX_ECS_NAME_L - 1)
     +             hdfattnms(PGSd_MET_NUM_OF_GROUPS)

C  Number of handles
      integer numhandles
      data             filen /'tblex6.hdf'/
      data             tbname /'Bolide Heights'/
      data             group /'Qi_group'/
      data             classname /'Fake Data class'/
      data             lat  /40.50, -22.81, 08.10/
      data             lon  /-80.22, -43.25, 98.32/
      data             height /400, 0, 0/
      data             numhandles/0/
      data  field /'Latitude(degrees),Longitude(degrees),Altitude(m)'/

C Map data buffer to data type-matched buffers
      EQUIVALENCE (data1(1), f1)
      EQUIVALENCE (data1(5), f2)
      EQUIVALENCE (data1(9),i3)
```

```
C     Set data type
      dtype = R32 //','// R32 //','// I32

      PRINT*,'*** Example6 ***'

C     Open file, using M-API parameter to define file access
      ier = OPMFIL(filen, CREATE_FILE, mfile)

      IF(IER.EQ.MAPIOK) THEN
         PRINT *,'Openned a Modis HDF file!'
      END IF

C     Create a Vgroup
      ier = CRMGRP(mfile,group,classname)

      IF(IER.EQ.MAPIOK) THEN
        PRINT *,'Created a Vgroup!'
      END IF

      if(ier.eq.MAPIOK) then
C       create an HDF table
        ier = CRMTBL(mfile,tbname,classname,group,field,dtype)
        IF(IER .EQ. MAPIOK) THEN
           PRINT *,'Successfully created a HDF table!'
        END IF

C       Put the data into the modis HDF table. Write 1 record
C       at a time, always append it at the end of the table(-1).
        recno = 1
        start = -1
        do 1 i =  1, 3
           f1 = lat(i)
           f2 = lon(i)
           i3 = height(i)
           if(ier.eq.MAPIOK) then
              ier = PMTBL(mfile,tbname,group,start,recno,data1)
           end if
    1   continue
        IF(IER .EQ. MAPIOK) THEN
           PRINT *,'Successfully wrote the table to MODIS HDF file!'
        END IF

c       complete the hdf file
        ier = CPMFIL(mfile, mdhandles, hdfattnms, numhandles)
        IF(IER .EQ. MAPIOK) THEN
           PRINT *,'MODIS HDF file was closed!'
        END IF

      end if

      print *,' '
      stop
      end
```

6.8 Example 7: Read HDF Tables in FORTRAN

This FORTRAN program demonstrates how to read an HDF table. The HDF file, "tblex6.hdf", (created by Example6) is openned for reading using OPMFIL. First, the information about the table is retrieved using GMFLDS. The actual table data is then retrieved using GMTBL. Once the table has been read into memory, the file is closed with a call to CLMFIL.

List of routines called:

Name	Description
OPMFIL	Opens a MODIS file (file access: r, w, a).
GMFLDS	Retrieves info on an HDF table.
GMTBL	Reads an HDF table into memory.
CLMFIL	Closes preexisitng MODIS file.

6.8.1 Source Code Listing for Example 7

```
C    The test program will first open the modis HDF table "Bolide Heights"
C    created by example6.f,  then call GMFLDS and GMTBL to get the
C    table's structural information and then the contents.

      program example7
      IMPLICIT NONE
      INCLUDE 'mapi.inc'

C MODIS file pointer array
      integer          mfile(MODFILLEN)
C Table name, data group name, Table's field names, field data types, and
class
      character*80     tbname, group, fldnm, dtype, classname
C maximum length of character strings returned by GMFLDS
      integer          strln
C Number of records in table, of fields (columns), of first record to read
      integer            recno, fldno, start
C Return code, type-matched buffer, size of read-in buffer
      integer            ret, height, bsize
C Type-matched buffers
      real*4           lat, lon
C Read-in data buffer
      byte             data(12)
      DATA tbname /'Bolide Heights'/
      DATA group  /'Qi_group'/
C Map data buffer to data type-matched buffers
      EQUIVALENCE (data(1), lat)
      EQUIVALENCE (data(5), lon)
```

```
      EQUIVALENCE (data(9), height)

      print*,'*** Example7 ***'
C     first open the HDF file.
      ret = OPMFIL("tblex6.hdf", "r", mfile)

      if (ret.eq.MAPIOK) then
C get the number of records and fields in the table, the table's class
C name, and the names of the fields and their respective data types.
         ret = GMFLDS(mfile, tbname, group, strln, recno,
     *              fldno, fldnm, dtype, classname)
         if (ret.eq.MAPIOK) then
            write(*,*) 'Record Numbers: ', recno
            write(*,*) 'Field Numbers: ', fldno
            write(*,*) 'Field Names: ', fldnm
            write(*,*) 'Data Types: ', dtype
            write(*,*) 'Classname: ', classname
         end if

C print the table contents, one record at a time
         write(*,*) 'Records:'
         do  start = 0, recno-1
            if (ret.eq.MAPIOK) then
               bsize = 12
               ret = GMTBL(mfile,tbname,group,fldnm,start,1,bsize,data)
               if (ret.eq.MAPIOK)
                  write(*,*)  lat, lon, height
               else
                 print*,'Error getting table row: ',start
            end if
          end do

C     close the HDF file.
          ret = CLMFIL(mfile)
          if(ret.ne.MAPIOK)then
             print*,'Error closing file.'
          else
            print*,'File closed.'
          endif
      endif
      print *,' '
      STOP
      END
```

6.9 Example 8: Read HDF Tables in C

This program performs the same operation as the FORTRAN program except that it is written in C. The HDF file created by Example 7 is opened for reading. A vgroup and table name are specified. The table information for "Bolide Heights" is retrieved. MODISsizeof is used to determine how much memory is needed to hold the table data. The data is retrieved and the file is closed. After the data has been retrieved, it is printed to the screen.

List of routines called:

Name	Description
openMODISfile	Opens a MODIS file (file access r, w, a).
getMODISfields	Retrieves HDF table info.
MODISsizeof	Determines size in bytes of an array type.
getMODIStable	Retrieves the HDF table data.
closeMODISfile	Closes a preexisiting MODIS file.

6.9.1 Source Code Listing for Example 8

```
/*  This example program demonstrates how to get a table's infor-
    mation by calling getMODISfields and how to get record(s) in
    a table by using getMODIStable. */

main()
{
  MODFILE   *modfile;         /* pointer to MODFILE structure */
  char filename[100] = "tblex6.hdf";      /* HDF file name */
  char tablename[256] = "Bolide Heights"; /* table name */
  char grpnm[256] = "Qi_group";           /* group name */
  long int stringlen = 256;         /* string length for either fieldnames or
                                       datatypes */
  long int recno;             /* number of records in a table */
  long int fieldno;                 /* number of fields in a table */
  char fieldnames[256];                   /* field names in a table */
  char datatypes[256];              /* data types in a table */
  char classname[256];              /* classname of the table */
  long int buf_size;                /* buffer size */
  long ier, i;
  unsigned char *data;              /* Data buffer */

  printf(" *** Example8 ***\n");
  /* Open the MODIS-HDF file */
```

```
  modfile = openMODISfile(filename, "r");
  if (modfile==NULL)
  {
    printf ("File not found\n");
    exit(-1);
  }
  else
    printf("File: %s opened.\n",filename);

  /* Get the table's information */
  ier = getMODISfields(modfile,tablename,grpnm,&stringlen,&recno,&fieldno,
                  fieldnames,datatypes,classname);
  printf("ier(getMODISfields) = %d\n",ier);
  printf("recno = %d\n",recno);
  printf("fieldno = %d\n",fieldno);
  printf("fieldnames = %s\n",fieldnames);
  printf("datatypes = %s\n",datatypes);
  printf("classname = %s\n",classname);

  /* Allocate memory for data */
  buf_size = recno * MODISsizeof(datatypes);
  data = (unsigned char *)malloc(buf_size);

  /* Get all the records in the table */
  ier =    getMODIStable(modfile,tablename,grpnm,
      fieldnames,0,recno,&buf_size,data);
  printf("ier(getMODIStable) = %d\n",ier);

  /* print out the retrived records */
  printf("data =\n");
  for (i=0; i<recno; i++)
    printf("%f %f %d\n",*((float *)(data + i * 12)),
           *((float *)(data + i * 12 + 4)),
           *((int32 *)(data + i * 12 + 8)) );

  /* Close the MODIS-HDF file */
  if ( closeMODISfile(&modfile) == MFAIL )
  {
    printf ("Error closing file\n");
    exit(1);
  }
  else
    printf ("File closed\n");

  printf("\n");
  exit(0);
}
```

6.10 Example 9: Read Data from ECS Metadata Files

This C program demonstrates how to read data from an ECS metadata file. (Note: there exists metadata incompatablities between PGS Toolkit version 5.0 and 5.1. The input file "metex9.hdf" supplied with the example programs was created using PGS Tool Kit version 5.1.) AN HDF file "metex9.hdf" is opened for reading using openMODISfile. The ECS metadata is retrieved from the HDF file using getMODISECSinfo. Once the metadata has been retrieved, it is then parsed into individual strings using sustrMODISECSinfo. The HDF file is closed using closeMODISfile.

List of routines called:

Name	Description
`openMODISfile`	Opens a MODIS file (file access r, w, a).
`getMODISECSinfo`	Retrieves the ECS metadata.
`MODISsizeof`	Determines size in bytes of an array type.
`getMODISfile`	Retrieves the value associated with an attribute.
`closeMODISfile`	Closes a MODIS file.

6.10.1 Source Code Listing for Example 9

```
/* End of example */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "mapi.h"

/* This example program demonstrates how to get MODIS ECS meta data by
   using getMODISECSinfo */

main() {
   MODFILE  *modfile;                    /* Modis file pointer */
   long int n_elements=256;                   /* Number of metadata values
                                     to extract from value. */
   void *value;
   char access_mode[]="r";
   int  ier;                                  /* Error code */
   int  i;
   long int  n_strings=10;
   char *substr[10];
   int  size = 256;

   char filename[]="metex9.hdf";              /* Input file */
```

```
/*  NOTE:  This file included is for use with PGS Toolkit v 5.1
    it is a documented fact that metadata incompatibilities
    exist between PGS Toolkit 5.0 and 5.1, in this case older
    files generated w/ 5.0 may not work  */
char PVLAttrName[]="CoreMetadata.0";       /* PVL Attribute name input */
char parmName[] = "SHORTNAME";              /* parameter name */
char data_type[] = "char *";               /* Data type off
                                  the parameter value */
printf(" *** Example 9 ***\n");
/* Allocate memory for value. */
value = (void *)malloc(size);

/* Open the MODIS-HDF file */
modfile= openMODISfile(filename, access_mode);

if (modfile==NULL)
{
  printf("Unable to open file %s\n",filename);
  exit(-1);
}
else
  printf("Opening the file\n");

/* Get MODIS ECS meta data */
ier = getMODISECSinfo(modfile, PVLAttrName, parmName, data_type,
               &n_elements, value);

/* Print output data */
printf("ier(getMODISECSinfo) = %d\n",ier);
printf("n_elements = %ld\n",n_elements);
printf("data_type = %s\n",data_type);

if ( (ier == MAPIOK) && ( n_elements != 0 ) ){
  if ( strcmp(data_type, I32) == 0 )
    for (i=0; i < n_elements; i++)
      printf("value = %ld ",((int32 *)value)[i]);
  if (strcmp(data_type, R32) == 0)
    for (i=0; i < n_elements; i++)
      printf("value = %f ",((float32 *)value)[i]);
  if (strcmp(data_type, R64) == 0)
    for (i=0; i < n_elements; i++)
      printf("value = %f ",((float64 *)value)[i]);
  printf("\n");
  if (strcmp(data_type, TXT) == 0)
  {
    ier = substrMODISECSinfo(value,n_elements,&n_strings,substr);
    if (ier==MFAIL)
    {
    printf("ier(substrMODISECSinfo) = %d\n",ier);
    printf("Error printing the substrings\n");
    }
    else
    {
    printf("n_strings = %d\n",n_strings);
```

```
       printf("string(s) = \n");
       for (i=0;i<n_strings;i++)
         printf("%s\n",substr[i]);
       }
     }
   }

   /* Close the MODIS-HDF file */
   ier = closeMODISfile(&modfile);
   if (ier == MFAIL)
   {
     printf("Error closing file\n");
     exit(-1);
   }
   else
   {
     printf("File closed.\n");
     exit(0);
   }

   printf("\n");
}
/* End of example */
```

APPENDIX A: ACRONYMS

ABI	Application Binary Interface
ANSI	American National Standards Institute
ASCII	American Standard for Computer Information Interchange
ATBD	Algorithm Theoretical Basis Document
AVHRR	Advanced Very High Resolution Radiometer
DAAC	Distributed Active Archive Center
DEC	Digital Equipment Corporation
DIF	Data Interchange Format
ECS	EOSDIS Core System
EOS	Earth Observing System
EOSDIS	Earth Observing System Data and Information System
ESDIS	Earth Science Data and Information System
FTP	File Transfer Protocol
GCMD	Global Change Master Directory
GSC	General Sciences Corporation
GSFC	Goddard Space Flight Center
HDF	Hierarchical Data Format
IDL	Interactive Data Language
I/O	Input/Output
IP	Internet Protocol
L1	Level 1
L1B	Level 1B
L2	Level 2
L3	Level 3
M-API	MODIS Applications Programming Interface
MCF	Metadata Configuration File
MODIS	Moderate Resolution Imaging Spectroradiometer
NASA	National Aeronautics and Space Administration
NCSA	National Center for Supercomputing Applications
ODL	Object Description Language

PCF	Process Control Files
PGE	Product Generation Executive
PVL	Parameter Value Language
QA	Quality Assurance
SAIC	Science Applications International Corporation
SCF	Science Computing Facilities
SD	Scientific Data
SDP	Science Data Processing
SDPS	Science Data Processing Segment
SDS	Scientific Data Set
SDST	Science Data Support Team
SeaWiFS	Sea-viewing Wide Field-of-view Sensor
SGI	Silicon Graphics, Inc.
SSTG	Science Software Transfer Group
STM	Science Team Member
TLCF	Team Leader Computing Facility
TRMM	Tropical Rainfall Measuring Mission
URLs	Uniform Resoure Locators
V	Vgroup
VS	Vdata Set
WWW	World Wide Web

APPENDIX B: M-API-SUPPLIED CONSTANTS AND MACROS

The following tables show the constants that are found in the mapi.h (C) and mapi.inc (FORTRAN):

Table B-1. SDS Metadata Constants

Metadata Description	Metadata Name	M-API Constant
array structure and dimension label string	"long_name"	MLONG_NAME
array structure and dimension units string	"units"	MUNITS
array structure and dimension format string	"format"	MFORMAT
array structure coordinate system string	"cordsys"	MCOORD_SYS
array structure Calibration factor	"scale_factor"	MSLOPE
array structure Calibration factor error	"scale_factor_err"	MSLOPE_ERROR
array structure uncalibrated offset	"add_offset"	MOFFSET
array structure uncalibrated offset error	"add_offset_err"	MOFFSET_ERROR
array structure uncalibrated data HDF number type	"calibrated_nt"	MNUM_TYPE
standard data valid range (Sdgetrange)[minimum,]	"valid_range"	MDATA_RANGE
array structure Fill Value	"_FillValue"	MFILL_VALUE
ECS inventory metadata global attribute name	"CoreMetadata.0"	MECS_CORE
ECS archive metadata global attribute name	"ProductMetadata.0"	MECS_ARCHIVE
'Same as above' - retained for Backward compatibility	"ProductMetadata.0"	MECS_PRODUCT

Table B-2. ECS Global Inventory Metadata Names

Note: User should refer to a particular file specification for a more precise layout of the metadata for a product.

Metadata Description	Metadata Name	M-API Constant
HDFattrNames = **MECS_CORE**		
References to all ancillary input files, (i.e., all input files other than MODIS products).	"ANCILLARYINPUTPOINTER"	MCORE_ANCIL_POINTER
Indicates the results of QA performed during product generation.	"AUTOMATICQUALITYFLAG"	MCORE_AUTO_QUALITY
Easternmost longitude of the granule spatial coverage.	"EASTBOUNDINGCOORDINATE"	MCORE_EAST_BOUND
Flag indicating whether points are on an inner (exclusion) G-ring.	"EXCLUSIONGRINGFLAG"	MCORE_EXCLUS_GRING_FLG
Self-reference to granule. For V1, this field should be identical to MODISPRODUCTFILENAME.	"GRANULEPOINTER"	MCORE_GRAN_POINTER
Latitudes of a series of points representing the perimeter of the granule spatial coverage (i.e., corners).	"GRINGPOINTLATITUDE"	MCORE_GRING_POINT_LAT
Longitudes of a series of points representing the perimeter of the granule spatial coverage.	"GRINGPOINTLONGITUDE"	MCORE_GRING_POINT_LON
Sequence numbers corresponding to perimeter latitudes and longitudes.	"GRINGPOINTSEQUENCENO"	MCORE_GRING_POINT_NUM
References to other MODIS product granules used as input for this product.	"INPUTPOINTER"	MCORE_INPUT_POINTER
A descriptive name for the data collection.	"LONGNAME"	MCORE_LONG_NAME
Northernmost latitude of the granule spatial coverage.	"NORTHBOUNDINGCOORDINATE"	MCORE_NORTH_BOUND
The granule level flag applying both generally to the granule and specifically to the parameters at the granule level. When applied to a parameter, the flag refers to the quality of that parameter in the granule.	"OPERATIONALQUALITYFLAG"	MCORE_OPER_QUAL_FLAG
Number of satellite orbit during which the granule data were collected.	"ORBITNUMBER"	MCORE_ORBIT_NUM
Reference to processing history file.	"PROCESSINGHISTORYPOINTER"	MCORE_HISTORY_POINTER
Value indicating the percent of interpolated data in the granule	"QAPERCENTINTERPOLATEDDATA"	MCORE_PERCENT_INTERP
Value indicating the percent of missing data in the granule.	"QAPERCENTMISSINGDATA"	MCORE_PERCENT_MISSING
Value indicating the percent of data in the granule outside of acceptable limits.	"QAPERCENTOUTOFBOUNDSDATA"	MCORE_PERCENT_OUT
A text explanation of the criteria used to set each quality flag; including thresholds or other criteria.	"QUALITYFLAGEXPLANATION"	MCORE_QUAL_EXPL
The date and time when the temporal coverage period of this granule began.	"RANGEBEGINNINGDATETIME"	MCORE_RANGE_START

Metadata Description	Metadata Name	M-API Constant
The date and time when the temporal coverage period of this granule ended.	"RANGEENDINGDATETIME"	MCORE_RANGE_END
Indicator of what reprocessing is planned for the granule.	"REPROCESSINGPLANNED"	MCORE_TO_BE_REDONE
Indicator of the reprocessing status of the granule.	"REPROCESSINGACTUAL"	MCORE_ACTUALLY_REDONE
The granule level flag applying to the granule and to the parameters at the granule level. When applied to a parameter, the flag refers to the quality of that parameter in the granule.	"SCIENCEQUALITYFLAG"	MCORE_SCIENCE_QUAL_FLG
The identifier for the data collection.	"SHORTNAME"	MCORE_SHORT_NAME
The size of the data granule in megabytes.	"SIZEMBECSDATAGRANULE"	MCORE_SIZE_OF_GRANULE
Southernmost latitude of the granule spatial coverage.	"SOUTHBOUNDINGCOORDINATE"	MCORE_SOUTH_BOUND
Westernmost longitude of the granule spatial coverage.	"WESTBOUNDINGCOORDINATE"	MCORE_WEST_BOUND
The MODIS filename for this granule.	"MODISPRODUCTFILENAME"	MPROD_FILENAME
MODIS mode of operation.	"OPERATIONMODE"	MPROD_OPERATIONMODE
This field contains the date and time the process that created this file was started.	"PROCESSINGDATETIME"	MPROD_PROC_DATE_TIME
The SPSO parameters for all data contained in this file, as listed in the SPSO database.	"SPSOPARAMETERS"	MPROD_SPSO_PARAM
The number of this MODIS granule.	"GRANULENUMBER"	MPROD_GRANULE_NUM
HDFattrNames = **MECS_PRODUCT**		
The date this algorithm package version successfully passed AI&T procedures and was accepted as an ECS standard algorithm.	"ALGORITHMPACKAGEACCEPTANCEDATE"	MPROD_ALGO_PCK_ACPT_DATE
This specifies the maturity of the algorithm package	"ALGORITHMPACKAGEMATURITYCODE"	MPROD_ALGO_PACK_MAT_CODE
Algorithm package name	"ALGORITHMPACKAGENAME"	MPROD_ALGO_PACK_NAME
The version of the algorithm package.	"ALGORITHMPACKAGEVERSION"	MPROD_ALGO_PACK_VER
The long name by which the instrument is known.	"INSTRUMENTNAME"	MPROD_INSTR_NAME
The short name assigned to the platform carrying the instrument.	"PLATFORMSHORTNAME"	MPROD_PLATFORM_SHORT_NAM
DAAC where product is processed.	"PROCESSINGCENTER"	MPROD_PROC_CENTER

Table B-3. Level 1A Macros

Metadata Description	Metadata Name	M-API Constant
MOD01_L1A	"MOD01_L1A"	MOD01_L1A
Scan number	"Scan number"	M01SCAN_NUMBER
Frame count array	"Frame count array"	M01FRAME_COUNT_ARRAY
Scan Type	"Scan Type"	M01SCAN_TYPE
SD start time	"SD start time"	M01SD_START_TIME
SRCA start time	"SRCA start time"	M01SRCA_START_TIME
BB start time	"BB start time"	M01BB_START_TIME
SV start time	"SV start time"	M01SV_START_TIME
EV start time	"EV start time"	M01EV_START_TIME
SRCA calibration mode	"SRCA calibration mode"	M01SRCA_CALIBRATION_MODE
Packet scan count	"Packet scan count"	M01PACKET_SCAN_COUNT
CCSDS Application Identifier	"CCSDS Application Identifier"	M01CCSDS_APID
Packet Quick Look flag	"Packet expedited data flag"	M01PACKET_QL
Mirror side	"Mirror side"	M01MIRROR_SIDE
Scan quality array	"Scan quality array"	M01SCAN_QUALITY_ARRAY
Earth sector Pixel quality	"Earth sector Pixel quality"	M01EV_PIX_QUAL
SD sector Pixel quality	"SD sector Pixel quality"	M01SD_PIX_QUAL
SRCA sector Pixel quality	"SRCA sector Pixel quality"	M01SRCA_PIX_QUAL
BB sector Pixel quality	"BB sector Pixel quality"	M01BB_PIX_QUAL
SV sector Pixel quality	"SV sector Pixel quality"	M01SV_PIX_QUAL
Bands 1 and 2	"EV_250m"	M01EV_250M
Bands 3 through 7	"EV_500m"	M01EV_500M
Bands 8 through 19	"EV_1km_day"	M01EV_1KM_DAY
Bands 20 through 36	"EV_1km_night"	M01EV_1KM_NITE
Bands 1 and 2	"SD_250m"	M01SD_250M
Bands 3 through 7	"SD_500m"	M01SD_500M
Bands 8 through 19	"SD_1km_day"	M01SD_1KM_DAY
Bands 20 through 36	"SD_1km_night"	M01SD_1KM_NITE
Bands 1 and 2	"SRCA_250m"	M01SRCA_250M
Bands 3 through 7	"SRCA_500m"	M01SRCA_500M
Bands 8 through 19	"SRCA_1km_day"	M01SRCA_1KM_DAY
Bands 20 through 36	"SRCA_1km_night"	M01SRCA_1KM_NITE
Bands 1 and 2	"BB_250m"	M01BB_250M
Bands 3 through 7	"BB_500m"	M01BB_500M
Bands 8 through 19	"BB_1km_day"	M01BB_1KM_DAY
Bands 20 through 36	"BB_1km_night"	M01BB_1KM_NITE
Bands 1 and 2	"SV_250m"	M01SV_250M
Bands 3 through 7	"SV_500m"	M01SV_500M
Bands 8 through 19	"SV_1km_day"	M01SV_1KM_DAY
Bands 20 through 36	"SV_1km_night"	M01SV_1KM_NITE
FPA DCR offset data	"fpa_dcr_offst"	M01FPA_DCR_OFFST
FAM Registration sample Delays	"fam_samp_delay"	M01FAM_SAMP_DELAY

Metadata Description	Metadata Name	M-API Constant
Raw mirror encoder data	"raw_mir_enc"	M01RAW_MIR_ENC
Current/Prior HK Telem	"raw_hk_telem"	M01RAW_HK_TELEM
Sci Eng Data	"raw_sci_eng"	M01RAW_SCI_ENG
Parameter Table	"raw_param"	M01RAW_PARAM
View Sector Start	"raw_vs_start"	M01RAW_VS_START
CP Event Log	"raw_cp_event"	M01RAW_CP_EVENT
FR Event Log	"raw_fr_event"	M01RAW_FR_EVENT
Raw s/c ancill data	"raw_sc_ancil"	M01RAW_SC_ANCIL
Dump Request Info	"raw_dump_req"	M01RAW_DUMP_REQ
Dump Data	"raw_dump_data"	M01RAW_DUMP_DATA
FPA/AEM Config	"fpa_aem_config"	M01FPA_AEM_CONFIG
FPA Use		M01FPA_USE

Table B-4. L1B/Geolocation Macros

Metadata Description	Metadata Name	M-API Constant
Product type identifier	"MOD02_L1B"	M02_PROD_ID
Software Version	"Software Version"	M02VERSION
Number of Scans	"Number of Scans"	M02NUMBER_OF_SCANS
Number of Day mode scans	"Number of Day mode scans"	M02NUMBER_OF_DAY_SCANS
Number of Night mode scans	"Number of Night mode scans"	M02NUMBER_OF_NIGHT_SCANS
Max Total Frames	"Max Total Frames"	M02MAX_TOTAL_FRAMES
Max Earth View Frames	"Max Earth Frames"	M02MAX_EARTH_FRAMES
Max SD Frames	"Max SD Frames"	M02MAX_SD_FRAMES
Max SRCA Frames	"Max SRCA Frames"	M02MAX_SRCA_FRAMES
Max BB Frames	"Max BB Frames"	M02MAX_BB_FRAMES
Max SV Frames	"Max SV Frames"	M02MAX_SV_FRAME
Scan types in product	"Scan types in product"	M02SCAN_TYPES
Dead MODIS Detectors	"Dead MODIS Detectors"	M02DEAD_DETECTORS
Noisy MODIS Detectors	"Noisy MODIS Detectors"	M02NOISY_DETECTORS
Dead Thermistors	"Dead Thermistors"	M02DEAD_THERMISTORS
Noisy Thermistors	"Noisy Thermistors"	M02NOISY_THERMISTORS
250 M Band Numbers for Reflected Solar Bands	"250 M Band Numbers for Reflected Solar Bands"	M02_250M_BAND_NUMS
500 M Band Numbers for Reflected Solar Bands	"500 M Band Numbers for Reflected Solar Bands"	M02_500M_BAND_NUMS
1000 M Band Numbers for Reflected Solar Bands	"1000 M Band Numbers for Reflected Solar Bands"	M02_1000M_REF_BAND_NUMS
Incomplete Scans	"Incomplete Scans"	M02PARTIAL_SCANS
Missing Packets	"Missing Packets"	M02MISSING_PACKETS
Packets with bad CRC	"Packets with bad CRC"	M02BAD_PACKETS
Discarded Packets	"Discarded Packets"	M02DISCARD_PACKETS
Swath Vgroup	"MODIS L1B Data"	M02SWATHWATH
num_scale_factors	"num_scale_factors"	M02NUM_SCALE_FACTORS
40*nscans	"40*nscans"	M02_40NSCANS
20*nscans	"20*nscans"	M02_20NSCANS
10*nscans	"10*nscans"	M02_10NSCANS
nscans	"nscans"	M02_NSCANS
40*nRefSBscans	M02_40NSCANS	M02_40NREFSBSCANS
20*nRefSBscans	M02_20NSCANS	M02_20NREFSBSCANS
10*nRefSBscans	M02_10NSCANS	M02_10NREFSBSCANS
Band_250M	"Band_250M"	M02BAND_250M
Band_500M	"Band_500M"	M02BAND_500M
Band_1KM_RefSB	"Band_1KM_RefSB"	M02BAND_1KM_REFSB
Band_1KM_Emissive	"Band_1KM_Emissive"	M02BAND_1KM_EMIS
4*BB frames	"4*BB_frames"	M02_4BB_FRAMES
2*BB frames	"2*BB_frames"	M02_2BB_FRAMES

Metadata Description	Metadata Name	M-API Constant
BB frames	"BB_frames"	M02_BB_FRAMES
4*EV frames	"4*EV_frames"	M02_4EV_FRAMES
2*EV frames	"2*EV_frames"	M02_2EV_FRAMES
EV frames	"EV_frames"	M02_EV_FRAMES
4*SD frames	"4*SD_frames"	M02_4SD_FRAMES
2*SD frames	"2*SD_frames"	M02_2SD_FRAMES
SD frames	"SD_frames"	M02_SD_FRAMES
4*SRCA frames	"4*SRCA_frames"	M02_4SRCA_FRAMES
2*SRCA frames	"2*SRCA_frames"	M02_2SRCA_FRAMES
SRCA frames	"SRCA_frames"	M02_SRCA_FRAMES
4*SV frames	"4*SV_frames"	M02_4SV_FRAMES
2*SV frames	"2*SV_frames"	M02_2SV_FRAMES
SV frames	"SV_frames"	M02_SV_FRAMES
Instrument Data Stored as Scientific Data Sets	"Slope_and_Offset"	M02SLOPE_AND_OFFSET
Black Body 250M Reflected Solar Bands Scaled Integer Radiance	"BB_250_RefSB_Rad"	M02BB_250
Black Body 250M Reflected Solar Bands Scaled Integer Radiance Uncertainty	"BB_250_RefSB_Rad_Uncert"	M02BB_250_UNCERT
Earth View 250M Reflected Solar Bands Scaled Integer Radiance	"EV_250_RefSB_Rad"	M02EARTH_RAD_250
Earth View 250M Reflected Solar Bands Scaled Integer Radiance Uncertainty	"EV_250_RefSB_Rad_Uncert"	M02EARTH_RAD_250_UNCERT
Solar Diffuser 250M Reflected Solar Bands Scaled Integer Radiance	"SD_250_RefSB_Rad"	M02DIFFUSER_250
Solar Diffuser 250M Reflected Solar Bands Scaled Integer Radiance Uncertainty	"SD_250_RefSB_Rad_Uncert"	M02DIFFUSER_250_UNCERT
RCA 250M Reflected Solar Bands Scaled Integer Radiance	"SRCA_250_RefSB_Rad"	M02SRCA_250
SRCA 250M Reflected Solar Bands Scaled Integer Radiance Uncertainty	"SRCA_250_RefSB_Rad_Uncert"	M02SRCA_250_UNCERT
Space View 250M Reflected Solar Bands Scaled Integer Radiance	"SV_250_RefSB_Rad"	M02SPACE_250
Space View 250M Reflected Solar Bands Scaled Integer Radiance Uncertainty	"SV_250_RefSB_Rad_Uncert"	M02SPACE_250_UNCERT
Black Body 500M Reflected Solar Bands Scaled Integer Radiance	"BB_500_RefSB_Rad"	M02BB_500
Black Body 500M Reflected Solar Bands Scaled Integer Radiance Uncertainty	"BB_500_RefSB_Rad_Uncert"	M02BB_500_UNCERT
Earth View 500M Reflected Solar Bands Scaled Integer Radiance	"EV_500_RefSB_Rad"	M02EARTH_RAD_500
Earth View 500M Reflected Solar Bands Scaled Integer Radiance Uncertainty	"EV_500_RefSB_Rad_Uncert"	M02EARTH_RAD_500_UNCERT
Solar Diffuser 500M Reflected Solar Bands Scaled Integer Radiance	"SD_500_RefSB_Rad"	M02DIFFUSER_500

Metadata Description	Metadata Name	M-API Constant
Solar Diffuser 500M Reflected Solar Bands Scaled Integer Radiance Uncertainty	"SD_500_RefSB_Rad_Uncert"	M02DIFFUSER_500_UNCERT
SRCA 500M Reflected Solar Bands Scaled Integer Radiance	"SRCA_500_RefSB_Rad"	M02SRCA_500
SRCA 500M Reflected Solar Bands Scaled Integer Radiance Uncertainty	"SRCA_500_RefSB_Rad_Uncert"	M02SRCA_500_UNCERT
Space View 500M Reflected Solar Bands Scaled Integer Radiance	"SV_500_RefSB_Rad"	M02SPACE_500
Space View 500M Reflected Solar Bands Scaled Integer Radiance Uncertainty	"SV_500_RefSB_Rad_Uncert"	M02SPACE_500_UNCERT
Black Body 1000M Reflected Solar Bands Scaled Integer Radiance	"BB_1000_RefSB_Rad"	M02BB_1000
Black Body 1000M Reflected Solar Bands Scaled Integer Radiance Uncertainty	"BB_1000_RefSB_Rad_Uncert"	M02BB_1000_UNCERT
Black Body 1000M Emissive Bands Scaled Integer Radiance	"BB_1000_Emissive"	M02BB_EMIS_1000
Black Body 1000M Emissive Bands Scaled Integer Radiance Uncertainty	"BB_1000_Emissive_Uncert"	M02BB_EMIS_1000_UNCERT
Earth View 1000M Reflected Solar Bands Scaled Integer Radiance	"EV_1000_RefSB_Rad"	M02EARTH_RAD_1000
Earth View 1000M Reflected Solar Bands Scaled Integer Radiance Uncertainty	"EV_1000_RefSB_Rad_Uncert"	M02EARTH_RAD_1000_UNCERT
Earth View 1000M Emissive Bands Scaled Integer Radiance	"EV_1000_Emissive_Rad"	M02EARTH_EMIS_RAD_1000
Earth View 1000M Emissive Bands Scaled Integer Radiance Uncertainty	"EV_1000_Emissive_Rad_Ucert"	M02EARTH_EMIS_RAD_1000_UNCERT
Solar Diffuser 1000M Reflected Solar Bands Scaled Integer Radiance	"SD_1000_RefSB_Rad"	M02DIFFUSER_1000
Solar Diffuser 1000M Reflected Solar Bands Scaled Integer Radiance Uncertainty	"SD_1000_RefSB_Rad_Uncert"	M02DIFFUSER_1000_UNCERT
Solar Diffuser 1000M Emissive Bands Scaled Integer Radiance	"SD_1000_Emissive_Rad"	M02DIFFUSER_EMIS_1000
Solar Diffuser 1000M Emissive Bands Scaled Integer Radiance Uncertainty	"SD_1000_Emissive_Rad_Ucert"	M02DIFFUSER_EMIS_1000_UNCERT
SRCA 1000M Reflected Solar Bands Scaled Integer Radiance	"SRCA_1000_RefSB_Rad"	M02SRCA_1000
SRCA 1000M Reflected Solar Bands Scaled Integer Radiance Uncertainty	"SRCA_1000_RefSB_Rad_Ucert"	M02SRCA_1000_UNCERT
SRCA 1000M Emissive Bands Scaled Integer Radiance	"SRCA_1000_Emissive_Rad"	M02SRCA_EMIS_1000
SRCA 1000M Emissive Bands Scaled Integer Radiance Uncertainty	"SRCA_1000_Emissive_Rad_Uncert"	M02SRCA_EMIS_1000_UNCERT
Space View 1000M Reflected Solar Bands Scaled Integer Radiance	"SV_1000_RefSB_Rad"	M02SPACE_1000
Space View 1000M Reflected Solar Bands Scaled Integer Radiance Uncertainty	"SV_1000_RefSB_Rad_Uncert"	M02SPACE_1000_UNCERT
Space View 1000M Emissive Bands Scaled Integer Radiance	"SV_1000_Emissive_Rad"	M02SPACE_EMIS_1000

Metadata Description	Metadata Name	M-API Constant
Space View 1000M Emissive Bands Scaled Integer Radiance Uncertainty	"SV_1000_Emissive_Rad_Uncert"	M02SPACE_EMIS_1000_UNCERT
Earth View 250M Reflected Solar Bands Scaled Integer Reflectance	"EV_250_RefSB_Refl"	M02EARTH_REFL_250
Earth View 250M Reflected Solar Bands Scaled Integer Reflectance Uncertainty	"EV_250_RefSB_Refl_Uncert"	M02EARTH_REFL_250_UNCERT
Earth View 500M Reflected Solar Bands Scaled Integer Reflectance	"EV_500_RefSB_Refl"	M02EARTH_REFL_500
Earth View 500M Reflected Solar Bands Scaled Integer Reflectance Uncertainty	"EV_500_RefSB_Refl_Uncert"	M02EARTH_REFL_500_UNCERT
Earth View 1000M Reflected Solar Bands Scaled Integer Reflectance	"EV_1000_RefSB_Refl"	M02EARTH_REFL_1000
Earth View 1000M Reflected Solar Bands Scaled Integer Reflectance Uncertainty	"EV_1000_RefSB_Refl_Uncert"	M02EARTH_REFL_1000_UNCERT
Eng. Packet 1 Data	"engineering_pkt_1"	M02ENG_PKT_1
Eng. Packet 2 Data	"engineering_pkt_2"	M02ENG_PKT_2
Mem. Packet 1 Data	"memory_pkt_1"	M02MEM_PKT_1
Mem. Packet 2 Data	"memory_pkt_2"	M02MEM_PKT_2
FPA DCR offset Data	"dcr_offset"	M02FPA_DCR_OFFSET
FAM Registration Sample Delays	"fam_sample_delay"	M02FAM_DELAY
Raw Mirror Encder Data	"mirror_encoder"	M02MIRROR_ENCODER
Current/Prior HK Telemtry	"hk_telemetry"	M02HK_TELEM
Science Engineering Data	"science_engineering"	M02SCI_ENG
Parameter Table	"parameter_table"	M02PARM_TABLE
View Sector Start	"view_vector_start"	M02VIEW_START
CP Event Log	"cp_event_log"	M02CP_LOG
FR Event Log	"fr_event_log"	M02FR_LOG
Raw S/C Ancillary Data	"spacecraft_ancillary_data"	M02SC_ANCIL
Dump Request Information	"dump_request_info"	M02DUMP_REQUEST
Dump Data	"dump_data"	M02DUMP
Instrument Telemetry	"instrument_telemetry"	M02INSTR_TELEM
Level 1B Swath Metadata Written as Vdata	"Level 1B Swath Metadata"	M02SWATH_MD
Scan Number /* I32 */	"Scan number"	M02SW_SCAN_NO
Total Frames /* I32 */	"Total frames"	M02SW_TOT_FRAMES
EV Frames /* I32 */	"EV frames"	M02SW_EV_FRAMES
SD Frames /* I32 */	"SD frames"	M02SW_SD_FRAMES
SRCA Frames /* I32 */	"SRCA Frames"	M02SW_SRCA_FRAMES
BB Frames /* I32 */	"BB Frames	M02SW_BB_FRAMES
SV Frames /* I32 */	"SV frames"	M02SW_SV_FRAMES
Scan Type /* TXT */	"Scan Type"	M02SW_SCAN_TYPE
Scan Start Time /* F64 */	"Scan start time"	M02SW_SCAN_START
Mirror Side /* I32 */	"Mirror Side"	M02SW_MIR_SIDE
Missing Packets /* I32 */	"Missing Packets"	M02SW_MISS_PKTS
Packets With Bad CRC /* I32 */	"Packets With Bad CRC"	M02SW_BAD_PKTS

Metadata Description	Metadata Name	M-API Constant
Discarded Packets /* I32 */	"Discarded Packets"	M02SW_DISC_PKTS
Moon in SV Port /* I32 */	"Moon in SV Port"	M02SW_MOON_OBS
On-Orbit Manuever /* TXT */	"On-Orbit Manuever"	M02SW_MANEUVER
No. SV Outliers /* I32 */	"No. SV Outliers"	M02SW_NUM_SV_OUTLIERS
No. BB Outliers /* I32 */	"No. BB Outliers"	M02SW_NUM_BB_OUTLIERS
No. thermistor outliers /* I32 */	"No. thermistor outliers"	M02SW_NUM_THERM_OUTLIERS
Product type identifier	"MOD03_Geolocation	M03_PROD_ID
Mirror axis error bias (gamma)	"gamma"	M03GAMMA
Nominal mirror rotation rate	"mir_rate"	M03MIR_RATE
Sample interval for 1 km bands	"t_frame"	M03T_FRAME
Mirror side 1 encoder-to-angle conversion coefficients (quadratic)	"poly_m1"	M03POLY_M1
Mirror side 2 encoder-to-angle conversion coefficients (quadratic)	"poly_m2"	M03POLY_M2
Spacecraft-to-instrument transformation matrix	"T_inst2sc"	M03T_INST2SC
Instrument-to-mirror transformation matrix	"T_mirr2inst"	M03T_MIRR2INST
Instrument-to-telescope transformation matrix	"T_tel2inst"	M03T_TEL2INST
Focal length for detectors (0 is ideal)	"Focal_length"	M03FOCAL_LENGTH
Band readout times relative to ideal band	"T_offset"	M03T_OFFSET
ECR orbit position at scan center time	"orb_pos"	M03ORB_POS
ECR orbit velocity at scan center time	"orb_vel"	M03ORB_VEL
ECR-to-instrument frame transformation matrix at scan center time	"T_inst2ECR"	M03T_INST2ECR
Spacecraft angular velocity in instrument frame	"ang_vel"	M03ANG_VEL
Unit Sun vector in ECR frame at scan center time	"sun_ref"	M03SUN_REF
Number of mirror encoder samples for this scan	"num_impulse"	M03NUM_IMPULSE
Mirror angles from encoder data	"impulse_enc"	M03IMPULSE_ENC
Mirror encoder sample times from start of scan	"impulse_time"	M03IMPULSE_TIME
Band-to-band geometric correction coefficients (based upon algorithm in ATBD)	"band_geo"	M03BAND_GEO
Geodetic longitude	"longitude"	M03LONGITUDE
Geodetic latitude	"latitude"	M03LATITUDE
Height above ellipsoid	"height"	M03HEIGHT
Sensor zenith	"SensorZenith"	M03SENSOR_ZEN
Sensor azimuth	"SensorAzimuth"	M03SENSOR_AZ
Range (pixel to sensor)	"Range"	M03RANGE
Solar zenith	"SolarZenith"	M03SOLAR_ZENITH

Metadata Description	Metadata Name	M-API Constant
Solar azimuth	"SolarAzimuth"	M03SOLAR_AZIMUTH
Geolocation flags	"gflags"	M03GFLAGS

Table B-5. Atmosphere Macros

Metadata Description	Metadata Name	Constant
MOD04_L2	"MOD04_L2"	M04L2_PROD_ID
MOD05_L2	"MOD05_L2"	M05L2_PROD_ID
MOD06_L2	"MOD06_L2"	M06L2_PROD_ID
MOD07_L2	"MOD07_L2"	M07L2_PROD_ID
MOD08_L2	"MOD08_L2"	M08L2_PROD_ID
MOD30_L2	"MOD30_L2"	M30L2_PROD_ID
MOD35_L2	"MOD35_L2"	M35L2_PROD_ID
MOD38_L2	"MOD38_L2"	M38L2_PROD_ID
1-km_Pixels_Per_Scan_Line	"1-km_pixels"	MAPIXELS_PER_SCAN
1-km_Scan_Lines_Per_Granule	"1-km_Scan_Lines_Per_Granule"	MALINES_PER_GRANULE
GMT Time of observation in milliseconds		MAGMT
Corner latitude of 10x10 pixel array	"Lat"	MACORNER_LAT
Corner longitude of 10x10 pixel array	"Lon"	MACORNER_LON
Scanline number through center of 5x5 pixel array	"Scanline_Number"	MASCANLINE_NO
Frame number of center pixel in 5x5 array		MAPIXEL_NO
Satellite zenith angle at midpoint of 5x5 array	"Sat_Zenith_Angle"	MAZENITH_SAT
Solar zenith angle at midpoint of 5x5 array	"Sun_Zenith_Angle"	MAZENITH_SOLAR
Index indicating the surface geography type as either Water(0) or Land(1)	"Land_Sea_Flag"	MAGEO_FLAG
Surface temperature at midpoint of 5x5 pixel array	"Sfc_Temp"	MATEMP_SFC
Surface pressure at midpoint of 5x5 pixel array	"Sfc_Pres"	MAPRES_SFC
Estimated tropopause height	"Height_Tropopause"	MATROPOPAUSE
long_name	"long_name"	MALONG_NAME
sampling_factor	"sampling_factor"	MASAMPLING
scale_factor	"scale_factor"	MASCALE
add_offset	"add_offset"	MAOFFSET
units	"units"	MAUNIT
valid_range	"valid_range"	MARANGE
Number Of Cells Across Swath	"Cells Across Swath"	MACELLS_ACROSS
Number Of Cells Along Swath	"Cells Along Swath"	MACELLS_ALONG
Pixels Per Scan Line	"Pixels Per Scan Line"	MAPIXELS
Number of Scan Lines	"Number of Scan Lines"	MASCANLINE
Number of Bands	"Number of Bands"	M04BANDS
Observed land reflectances averaged on 10x10 1-km pixel array	"Avg_Refl"	M04LAND_REFLS
Land aerosol optical thickness (AOT) for continental model	"Opt_Thickness_M1"	M04LAND_OPT_THICK
Standard deviation of observed land reflectances	"Std_Dev_Refl"	M04LAND_REFLS_DEV

Metadata Description	Metadata Name	Constant
Land AOT for corrected model	"Opt_Thickness_M2"	M04LAND_OPT_THICK_COR
Aerosol path radiance ratio (continental model) of red to blue channel (band 3/band 1)	"Aerosol_Path_Rad_Ratio"	M04LAND_RADIANCE_RATIO
Relative contribution of smoke/sulfate particles to dust in the computation of the aerosol optical depth	"Relative_Contribution"	M04LAND_CONTRIBUTION
Number of Clear Land Pixels in Band 3	"Number_of_Pixels_B3"	M04LAND_PIXELS_B3
Number of Clear Land Pixels in Band 1	"Number_of_Pixels_B1"	M04LAND_PIXELS_B1
Identification of retrieval procedure	"Procedure_ID"	M04LAND_PROC_ID
Aerosol type in one of four categories: continental, dust, sulfate, and smoke	"Aerosol_Type"	M04LAND_AERO_TYPE
Aerosol land error flag	"Error_Flag"	M04LAND_ERROR
Ocean AOT at 0.55 micron on 10x10 1-km pixel array	"Opt_Thickness"	M04OCEAN_OPT_THICK
Small-particle ocean AOT at 0.55 micron on 10x10 pixel array	"Opt_Thickness_Small"	M04OCEAN_OPT_THICK_S
Large-particle ocean AOT at 0.55 micron on 10x10 pixel array	"Opt_Thickness_Large"	M04OCEAN_OPT_THICK_L
Weight factor for combining large and small aerosol modes during retrieval. This parameter minimizes the least-squares error summed over spectral bands	"Error_Min_Factor"	M04OCEAN_ERROR
Solution number from 1 to 36	"Solution_Number"	M04OCEAN_SOLUTION
Observed ocean reflectances averaged on 10x10 1-km pixel array	"Avg_Refl"	M04OCEAN_REFLS
Look-Up Table of Aerosol Model Parameters and Values Vdata	"LUT_Data"	M04AEROSOL_LUT
small mode aerosol mean radius	"RGSS"	M04LUT_RGSS
large mode aerosol mean radius	"RGSB"	M04LUT_RGSB
standard deviation of small mode radius	"SIGMAS"	M04LUT_SIGMAS
standard deviation of large mode radius	"SIGMAB"	M04LUT_SIGMAB
CCN	"CNNS"	M04LUT_CCNS
small mode extinction coefficient for 5 wavelengths	"EXTS"	M04LUT_EXTS
large mode extinction coefficient for 5 wavelengths	"EXTB"	M04LUT_EXTB
moments order 1-4 of small mode particle radius	"MOMENTS"	M04LUT_MOMENTS
moments order 1-4 of large mode particle radius	"MOMENTB"	M04LUT_MOMENTB
small mode backscatter ratio for 5 wavelengths	"BACKSCTS"	M04LUT_BACKSCTS
large mode backscatter ratio for 5 wavelengths	"BACKSCTB"	M04LUT_BACKSCTB
small mode asymmetry factor for 5 wavelengths	"ASSYMS"	M04LUT_ASSYMS
large mode asymmetry factor for 5 wavelengths	"ASSYMB"	M04LUT_ASSYMB

Metadata Description	Metadata Name	Constant
small mode albedo for 5 wavelengths	"ALBEDOS"	M04LUT_ALBEDOS
large mode albedo for 5 wavelengths	"ALBEDOB"	M04LUT_ALBEDOB
Total column water vapor amounts over clear land, and cloud scenes over land and ocean	"Column_Water_Vapor"	M05WATER_VAPOR
Index indicating cloud(0), no cloud(1), or cloud/no cloud determination not made(-1)	"Cloud_Qualifier"	M05CLOUD_QUAL
Number_Of_1-km_Bands	"Number_Of_1-km_Bands "	M06BANDS
Number of Channel Indices	"Number of Channel Indices"	M06CHANNEL_IND
Number of Channel Differences	"Number of Channel Differences"	M06CHANNEL_DIFF
Brightness temperatures for IR channels 27 - 36 at 5x5 1-km pixel resolution	"Brightness_Temp"	M06BRIGHT_TEMP
Sufficient number of cloudy pixels (0) or too few cloudy pixels (1) to be able to process 5x5 pixel array	"Processing_Flag"	M06PROCESS_FLAG
Spectral cloud forcing for IR channels 29, and 31 - 36	"Spec_Cloud_Forcing"	M06CLOUD_FORCING
value to indicate the method of cloud height determination	"Cloud_H_Method"	M06METHOD
Cloud top effective emissivity	"Cloudtop_Eff_Emi"	M06EMISSIVITY_CT
Cloud top pressure	"Cloudtop_Pres"	M06PRES_CT
Cloud top temperature	"Cloudtop_Temp"	M06TEMP_CT
Cloud fraction at 5x5 1-km pixel resolution	"Cloud_Fraction"	M06FRACTION
Separate cloud top pressure estimates from five radiances ratios	"Cloudtop_Pres_From_Ratios"	M06PRES_CT_RATIO
Cloud top pressure from IR window	"Cloudtop_Pres_IR"	M06PRES_CT_IR
Surface type index	"Sfc_Type"	M06SFC_TYPE
Radiance variance for channels 29, 31, and 32	"Radiance_Var"	M06RADIANCE
Brightness temperature differences between IR channels 29, 31, and 32	"Brightness_Temp_Diff"	M06BRIGHT_TEMP_DIFF
Cloud thermodynamic phase derived from infrared retrieval algorithm	"Cloud_Phase_IR"	M06PHASE_IR
Effective particle radius at 1-km resolution	"Eff_Particle_Rad"	M06EFF_RADIUS
Cloud optical thickness at 1-km pixel resolution	"Cloud_Opt_Thickness"	M06CLOUD_OPT_THICK
Cloud thermodynamic phase derived from visible/SW infrared retrieval algorithm	"Cloud_Phase_VIS"	M06PHASE_VIS
Statistics at 1-km pixel resolution	"Statistics"	M06STATISTICS
Total Column Ozone at 5x5 1-km pixel resolution	"Total_Ozone"	M07TOTAL_OZONE
Total Totals Atmospheric Stability Index	"Total_Totals"	M08TOTALS
Lifted Index Atmospheric Stability Index	"Lifted_Index"	M08LIFTED_INDEX
K Index Atmospheric Stability Index	"K_Index"	M08K_INDEX

Metadata Description	Metadata Name	Constant
Number Of Channels		M30CHANNELS
Brightness temperatures for IR channels 20, 22-25, and 27-36	"Brightness_Temp"	M30BRIGHT_TEMP
Guess temperature profile for 20 vertical levels	"Guess_Temp_Profile"	M30TEMP_PROF
Guess dewpoint temperature profile for 15 vertical levels	"Guess_DewP_Profile"	M30DEWP_TEMP_PROF
Rretrieved temperature profile for 20 vertical levels	"Retr_Temp_Profile"	M30RETR_TEMP_PROF
Retrieved dewpoint temperature profile for 15 vertical levels	"Retr_DewP_Profile"	M30RETR_DEWP_TEMP_PROF
Index of pressure levels for the 15 vertical levels	"Index_Of_Pressure_Levels"	M30PRESS_LEVEL
Bit field mask containing the results of visible and infrared radiance cloud/no cloud tests	"Cloud_Mask"	M35CLOUD_MASK
Cell Frame Number	"Cell Frame Number"	M38CELL_FRAME
Cell Line Number	"Cell Line Number"	M38CELL_LINE
Atmospheric Water Vapor Parameter at 5x5 1-km pixel resolution	"Water Vapor"	M38WATER_VAPOR

Table B-6. Ocean Macros

Metadata Description	Metadata Name	M-API Constant
MOD27 HDF output file	"MOD27 HDF output file"	M27_PROD_ID
output_file_name	"output_file_name"	M27O_F_NAME
output_file_logical file number	"output_file_logical file number"	M27O_F_L_F_NUM
units_of_output_file_logical_file_number	"units_of_output_file_logical_file_number"	U_O_O_F_L_F_NUM
product_name	"product_name"	M27P_NAME
statistics_file_name	"statistics_file_name"	M27S_F_NAME
product_sum_total_over_all_regions	"product_sum_total_over_all_regions"	M27P_SUM
units_of_product_sum_total_over_all_regions	"units_of_product_sum_total_over_all_regions"	M27U_O_P_SUM
product_variance_total_over_all_regions	"product_variance_total_over_all_regions"	M27P_VAR
units_of_product_variance_total_over_all_regions	"units_of_product_variance_total_over_all_regions"	M27P_O_P_VAR
product_area_total_over_all_regions	"product_area_total_over_all_regions"	M27P_AREA
units_of_product_area_total_over_all_regions	"units_of_product_area_total_over_all_regions"	M27U_O_P_AREA
square km	"square km"	M27SQKM
number_of_regions_for_product	"number_of_regions_for_product"	M27P_NREGS
coordinate_system	"coordinate_system"	M27COORD_SYS
units_of_coordinate_system	"units_of_coordinate_system"	M27U_O_COORD_SYS
range_of_coordinate_system	"range_of_coordinate_system"	M27R_O_COORD_SYS
character_counter	"character_counter"	M27KCHAR
region_counter	"region_counter"	M27JREG
limit_of_region_counter	"limit_of_region_counter"	M27KLIM
function_order_counter	"function_order_counter"	M27KORD
product_cell_counter	"product_cell_counter"	M27KCELLS
name_of_regions	"name_of_regions"	M27NAME_R
limit_of_regions-deg_lat_and_deg_long	"limit_of_regions-deg_lat_and_deg_long"	M27LIM_R
area_of_regions-km_squared	"area_of_regions-km_squared"	M27AREA_R
independent_variables_of_regions	"independent_variables_of_regions"	M27IV_R
functions_used_in_regions	"functions_used_in_regions"	M27FUNCTIONS_R
order_of_functions_used_in_regions	"order_of_functions_used_in_regions"	M27ORD_R
coefficients_used_in_regions	"coefficients_used_in_regions"	M27COEFF_R
error_in_regions-gr_per_m3_per year	"error_in_regions-gr_per_m3_per year"	M27ERR_R
sum_in_regions-gr_per_m3_per_year	"sum_in_regions-gr_per_m3_per_year"	M27SUM_R
variance_in_regions-gr2_per_m6_per_year2	"variance_in_regions-gr2_per_m6_per_year2"	M27VAR_R
product_y-gr_per_m3_per_year	"product_y-gr_per_m3_per_year"	M27P_Y
product_error_ey-gr_per_m3_per_year	"product_error_ey-gr_per_m3_per_year"	M27P_EY

Table B-7. Land Macros

Metadata Description	Metadata Name	M-API Constant
Pixels_per_scan_line	"Pixels_per_scan_line"	MLPIXELS_PER_SCAN
Number_of_scan_lines	"Number_of_scan_lines"	MLNUMBER_OF_LINES
Pixels_per_line	"Pixels_per_line"	MLPIXELS_PER_LINE
Lines_per_tile	"Lines_per_tile"	MLLINES_PER_TILE
Total_observations	"Total Observations"	MLTOTAL_OBSERVATIONS
Num_parameters	"Num_parameters"	MLNUMBER_OF_PARAMS
Maximum_observations	"Maximum Observations"	MLMAX_OBSERVATIONS
Number_of_granules	"Number of Granules"	MLNUMBER_OF_GRANULES
Granule_IDs	"Granule_IDs"	MLGRANULE_IDS
File_Format	"L2G Storage Format"	MLFILE_FORMAT
Parameter1	"Parameter1"	MLPARM1
Parameter2	"Parameter2"	MLPARM2
Parameter3	"Parameter3"	MLPARM3
Parameter4	"Parameter4"	MLPARM4
Parameter5	"Parameter5"	MLPARM5
Parameter6	"Parameter6"	MLPARM6
Parameter7	"Parameter7"	MLPARM7
Year	"Year"	MLYEAR
Day_of_year	"Day_of_year"	MLDOY
nrow	"nrow"	MLNUMBER_OF_ROWS
nest_lev	"Grid Nesting Level"	MLNEST_LEVEL
ref_lon_in_deg	"ref_lon_in_deg"	MLREF_LONGITUDE
ang_size_in_arcsec	"Characteristic Bin Angular Dimension"	MLANGULAR_SIZE
irow_start	"irow_start"	MLIROW_START
ncol_max	"ncol_max"	MLNCOL_MAX
itile_horiz	"itile_horiz"	MLITILE_HORIZ
itile_vert	"itile_vert"	MLITILE_VERT
ntile_horiz	"ntile_horiz"	MLNTILE_HORIZ
ntile_vert	"ntile_vert"	MLNTILE_VERT
L2G number of observations per pixel contained within L2G file	"num_observations"	MLNUMBER_OF_OBS
The number of columns in the full ISCCP grid for each row (line) contained within the L2G file	"ncol"	MLNUMBER_OF_COLS
The start column in the full ISCCP grid for each row (line) contained within the L2G file (starting at zero).	"icol_start"	MLSTART_COLUMN
The number of columns in each row (line) contained within the L2G file.	"ncol_tile"	MLCOLS_PER_ROW

Metadata Description	Metadata Name	M-API Constant
The start pixel of the first valid column in each row (line) contained within the L2G file (starting at zero).	"ipix_start"	MLSTART_PIX
Number of observations per line	"nobs_line"	MLOBS_PER_LINE
SPSO_parameter	"SPSO_parameter"	MLSPSO_PARAMETERS
Product type identifier: MOD09_ANG_L2G_1KM	"MOD.AM1.geoang.L2G"	M09ANG_PROD_ID
Zenith angle to sensor	"SensorZenith"	M09SENSOR_ZENITH
Azimuth angle to sensor	"SensorAzimuth"	M09SENSOR_AZIMUTH
Distance to sensor	"Range"	M09SENSOR_DISTANCE
Zenith angle to sun	"SolarZenith"	M09SOLAR_ZENITH
Azimuth angle to sun	"SolarAzimuth"	M09SOLAR_AZIMUTH
Product type identifier: MOD09_PNT_L2G_1KM	"MOD.AM1.pntr_1km.L2G"	M09PNT1K_PROD_ID
Product type identifier: MOD09_PNT_L2G_500M	"MOD.AM1.pntr_500m.L2G"	M09PNT500_PROD_ID
Product type identifier: MOD09_PNT_L2G_250M i	"MOD.AM1.pntr_250m.L2G"	M09PNT250_PROD_ID
Pointer to granule IDs from which the observation came. Zero relative. Fill value is 255.	"granule_pnt"	M09GRANULE_PNT
Sample number of observation (1 km spatial element) in granule	"sample"	M09OBS_IN_GRANULE
Sub-pixel (delta) line location of cell center in observation footprint. Relative to center of observation specified by (line, sample).	"dline"	M09CELL_CENTER
Sub-pixel (delta) line location of cell center in observation footprint SDS. Relative to center of observation specified by (line, sample).	"dsample"	M09SAMPLE_CENTER
Observation coverage SDS: area of intersection between observation footprint and cell divided by area of observation.	"obscov"	M09OBS_COVERAGE
Cell coverage SDS: area of intersection between observation footprint and cell divided by area of cell.	"cellcov"	M09CELL_COVERAGE
Product type identifier: MOD09_L2 and MOD13_L2	"MOD.AM1.V1.srefl_500m.L2G"	M09_L2G_500M_PROD_ID
Surface Reflectance for MODIS Band 3	"sur_refl_b03"	M09BAND3_SURF_REFL
Surface Reflectance for MODIS Band 4	"sur_refl_b04"	M09BAND4_SURF_REFL
Surface Reflectance for MODIS Band 5	"sur_refl_b05"	M09BAND5_SURF_REFL
Surface Reflectance for MODIS Band 6	"sur_refl_b06"	M09BAND6_SURF_REFL
Surface Reflectance for MODIS Band 7	"sur_refl_b07"	M09BAND7_SURF_REFL

Metadata Description	Metadata Name	M-API Constant
Indicators of the quality of the 500 m reflectance data	"QC_500m"	M09QUALITY_500
Product type identifier: MOD09_L2 and MOD13_L2	"MOD.AM1.V1.srefl_250m.L2G"	M09_L2G_250M_PROD_ID
Surface Reflectance for MODIS Band 1	"sur_refl_b01"	M09BAND1_SURF_REFL
Surface Reflectance for MODIS Band 2	"sur_refl_b02"	M09BAND2_SURF_REFL
Indicators of the quality of the 250 m reflectance and VI data integrity.	"QC_250m"	M09QUALITY_250
Product type identifier: MOD09_L2 and MOD13_L2 MOD09SUBS_L2G_16DY	"MOD.AM1.brdfsubs.L3"	M09_REFLDB_PROD_ID
ang_size (in arcsec)	"ang_size (in arcsec)"	M09_REFLDB_ANGULAR_SIZE
General information on observational basis M09_OBS_INFO words	"Obs_Info_Items"	M09_OBS_INFO_WORDS
Viewing and illumination angles	"Angles"	M09_ANGLES
N_obs_dy	"Num_Obs_Max"	M09_ANGLES_OBS
N_angles	"Num_Angles"	M09_ANGLES_NUM
Surface reflectances	"Surface_Refl"	M09_REFLDB_SURF_REFL
N_obs_dy	"Num_Obs_Max"	M09_SURF_REFL_OBS
N_bands	"Num_Land_Bands"	M09_SURF_REFL_BANDS
Quality and weights of the respective observations	"Weights_QC"	M09_QUALITY_WEIGHTS
N_obs_dy	"Num_Obs_Max"	M09_QUALITY_OBS
words	"Num_Weights_QC"	M09_QUALITY_WORDS
Product type identifier: MOD09_BARS	"MOD.AMI.bars_16dy.L3"	M09BARS_PROD_ID
Nadir-equivalent surface reflectances for MODIS bands 1-7	"BARS"	M09BARS
Overall quality of the BRDF-adjusted surface reflectances	"BARS_QC"	M09BARS_QC
The number of columns in the full ISCCP grid for each row (line) contained within this L2G file.	"ncols"	M09NCOL
The start column in the full ISCCP grid for each row (line) contained within this L2G file (starting at zero).	"icol_start"	M09ICOL_START
The number of columns in each row (line) contained within this L2G file.	"ncol_tile"	M09NCOL_TILE
The start pixel of the first valid column in each row (line) contained within this L2G file (starting at zero).	"ipix_start"	M09IPIX_START
Product type identifier: MOD09_L3_16DY_G	"MOD_AM1.brdf_16dy.L3"	M09_L3_PROD_ID
Identifier for BRDF models chosen	"BRDF_Model_ID"	M09BRDF_MODEL_ID
RMSE for BRDF models chosen	"BRDF_Model_RMSE"	M09BRDF_MODEL_RMSE
BRDF quality control	"Quality_Control"	M09QUALITY

Metadata Description	Metadata Name	M-API Constant
BRDF parameters for the seven land bands	"BRDF_Parameters"	M09BRDF_PARAMETERS
Albedo parameters for broadband, < 0.7 mu-m, > 0.7 mu-m, and the seven land bands.	"Albedo"	M09ALBEDO
A neasure of fit from RMSE and sampling of all models tested.	"Fit_Assessments"	M09FIT_ASSESS
The number of columns in the full ISCCP grid for each row (line) contained within this L3 file.	"ncol"	M09NCOL
The start column in the full ISCCP frid for each row (line) contained within this L3 file (starting at zero).	"icol_start"	M09ICOL_START
The number of columns in each row (line) contained within this L3 file.	"ncol_tile"	M09NCOL_TILE
The start pixel of the first valid column in each row (line) contained within this L3 file (starting at zero).	"ipix_start"	M09IPIX_START
N_select_models	"N_select_models"	M09N_SELECT_MODELS
words	"words"	M09WORDS
land_bands	"Num_Land_Bands"	M09LAND_BANDS
number_parameters	"Num_BRDF_Parameters"	M09NUMBER_PARAMETERS
land_bands_and_broadband_and_<>_0.7mu-m	"land_bands_and_broadband_and_<>_0.7mu-m"	M09LANDBANDS_BROADBAND_OTHER
N_models	"N_models"	M09N_MODELS
Product type identifier: MOD09_L2 and MOD13_L2	MOD.AM1.srefl.L2	M0913_L2_PROD_ID
SurfaceReflectance for MODIS Band 1 SDS	"sur_refl.b01"	M0913BAND1_SURF_REFL
SurfaceReflectance for MODIS Band 2 SDS	"sur_refl.b02"	M0913BAND2_SURF_REFL
SurfaceReflectance for MODIS Band 3 SDS	"sur_refl.b03"	M0913BAND3_SURF_REFL
SurfaceReflectance for MODIS Band 4 SDS	"sur_refl.b04"	M0913BAND4_SURF_REFL
SurfaceReflectance for MODIS Band 5 SDS	"sur_refl.b05"	M0913BAND5_SURF_REFL
SurfaceReflectance for MODIS Band 6 SDS	"sur_refl.b06"	M0913BAND6_SURF_REFL
SurfaceReflectance for MODIS Band 7 SDS	"sur_refl.b07"	M0913BAND7_SURF_REFL
NDVI index at 250m	"NDVI_index"	M0913_NDVI_INDEX
MVI index at 250m	"MVI_index"	M0913_MVI_INDEX
Indicators of the quality of the 250m reflectance and VI data integrity.	"QC_250m"	M0913QUALITY_250

Metadata Description	Metadata Name	M-API Constant
Indicators of the quality of the 500m reflectance and VI data integrity.	"QC_500m"	M0913QUALITY_500
num_detectors	"num_detectors"	M0913NUM_DETECTORS
sampling	"sampling"	M0913SAMPLING
Number_of_pixels_processed	"Number_of_pixels_processed"	M10PROCESSED_PIXELS
Total_snow_pixels	"Total_snow_pixels	M10SNOW_PIXELS
Percentage_snow	"Percentage_snow"	M10PERCENT_SNOW
Percentage_not_snow	"Percentage_not_snow"	M10PERCENT_NOT_SNOW
Above_range_NDSI	"Above_range_NDSI"	M10NDSI_ABOVE
Below_range_NDSI	"Below_range_NDSI"	M10NDSI_BELOW
Division_by_zero	"Division_by_zero"	M10ZERO_DIVIDE
Out_of_range_input	"Out_of_range_input"	M10OUT_OF_RANGE_INPUT
No_decision	"No_decision"	M10NO_DECISION
L2/L2G Identification of daily snow cover on the land surface	"daily_snow_cover"	M10DAILY_SNOW
Product type identifier: MOD10_L2G	"MOD.AM1.V1.snow.L2G"	M10L2G_PROD_ID
Product type identifier: MOD10_L3_DY_G	"MOD.AM1.V1.snow_dy.L3"	M10L3_PROD_ID
L3 Identification of daily snow cover on the land surface	"Daily_Gridded_Snow_Cover"	M10GRIDDED_SNOW
Product type identifier: MOD11_L2	"MOD.AM1.V1.lst.L2"	M11L2_PROD_ID
L2/L2G Identification of Land Surface Temperature	"LST"	M11SURF_TEMP
L2/L2G LST Quality Indicator	"QC"	M11QUALITY
L2/L2G Error in land surface temperature measurements	"Error_LST"	M11ERRORS
L2/L2G/L3 Band 31 emissivity	"Emis_31"	M11BAND31_EMIS
L2/L2G/L3 Band 32 emissivity	"Emis_32"	M11BAND32_EMIS
L2/L2G Band 29 or band 20 emissivity	"Emis_29"	M11BAND29OR20_EMIS
Product type identifier: MOD11_L2G	"MOD11_L2G"	M11L2G_PROD_ID
Product type identifier: MOD11_L3_WK_G	"MOD.AM1.V1.lst_1dy_cmg.L3"	M11L3_1DY_PROD_ID
L3 Identification of Land Surface Temperature	"LST"	M11L3SURF_TEMP
Land surface temperature in view within 45deg	"LST_view<45d"	M11NARROW_LST
L3 LST Quality Indicator	"QC"	M11L3QUALITY
Land-Surface Temperature Standard Deviation	"Stdv_LST"	M11STD_DEV
L3 Band 29 or band 20 emissivity	"Emis_29"	M11L3BAND29OR20_EMIS
Angular coefficients for Band 31 emissivity	"Ang_Coef_Emis_31"	M11BAND31_ANG_COEFS
Angular coefficients for Band 32 emissivity	"Ang_Coef_Emis_32"	M11BAND32_ANG_COEFS
Product type identifier: MOD12_L3_3MN_D/MOD12_L3_3MN_F	"MOD.AM1.V1.lc_1km.L3.3m"	M12L3_PROD_ID

Metadata Description	Metadata Name	M-API Constant
ang_size (in arcsec)	"ang_size (in arcsec)"	M12ANGULAR_SIZE
Identification of land cover type	"Land Cover Type"	M12LAND_COVER
Identification of Overall quality of the land cover	"Type Overall QC"	M12QUALITY
Identification of Number of products generated since last classification update	"Num Product Gen"	M12PRODS_GENERATED
Identification of Number of snow months over pervious 12 months	"Snow Months"	M12SNOW_MONTHS
Identification of Number of BRDFs used for classification that have been derived within the pass 12 month	"Num BRDF"	M12BRDFS_USED
Identification of Confidence in BRDF/reflectance correction	"BRDF Internal QC"	M12BRDF_STOCK
Identification of Number of LST values used for classification	"Num LST"	M12LST_VALS_USED
Identification of Confidence in VI over 12 months	"VI Internal QC"	M12VI_STOCK
Identification of TBD quality control for land cover type	"Land_cover_TBD_1"	M12QUALITY1
Identification of TBD quality control for land cover type	"Land_cover_TBD_2"	M12QUALITY2
Identification of Land cover change	"Land Cover Change"	M12LAND_COVER_CHANGE
Identification of Quality control for land cover change	"Land Cover Change QC"	M12CHANGE_QUALITY
Product type identifier: MOD14_L2	"MOD14_L2"	M14L2_PROD_ID
L2/L2G Identification of fire on the land surface	"fire_mask"	M14LAND_FIRE
L2/L2G/L3 Total emmitted energy detected	"power"	M14ENERGY
L2/L2G/L3 Class of fire detected	"smold"	M14FIRE_CLASS
Fire quality control	"fire_qc"	M14QUALITY
Product type identifier: MOD14_L2G	"MOD.AM1.V1.fire.L2G"	M14L2G_PROD_ID
L2G/L3 Fire quality control	"fire_qc"	M14L2GQUALITY
Product type identifier: MOD14_L3	"MOD.AM1.V1.fire_daily.L3"	M14L3_PROD_ID
Product type identifier: MOD29_L2	"MOD.AM1.V1.seaice_max.L2"	M29L2_PROD_ID
Total_sea_ice_pixels	"Total_sea_ice_pixels"	M29SEA_ICE_PIXELS
Percentage_sea_ice	"Percentage_sea_ice"	M29SEA_ICE_PERCENT
Percentage_not_sea_ice	"Percentage_not_sea_ice"	M29NOT_SEA_ICE_PERCENT
Above_range_NDSI	"Above_range_NDSI"	M29NDSI_ABOVE
Below_range_NDSI	"Below_range_NDSI"	M29NDSI_BELOW
Division_by_zero	"Division_by_zero"	M29ZERO_DIVIDE
Out_of_range_input	"Out_of_range_input"	M29OUT_OF_RANGE
No_decision	"No_decision"	M29NO_DECISION
Identification of daily sea ice cover	"daily_sea_ice_cover"	M29DAILY_SEA_ICE
Product type identifier: MOD29_L2G	"MOD29_L2G"	M29L2G_PROD_ID

Metadata Description	Metadata Name	M-API Constant
Daily Ice Cover	"daily_ice_cover	M29L2GDAILY_SEA_ICE
Product type identifier: MOD29_L3_DY_G	"MOD.AM1.seaice_max_dy.L3"	M29L3_PROD_ID
Identification of daily sea ice cover	"daily_gridded_sea_ice_cover"	M29L3DAILY_SEA_ICE
Product type identifier: MOD33_L3_WK_G	"MOD.AM1.V1.snow_10dy.L3"	M33L3_PROD_ID
Weekly Snow Cover	"Composite_Snow_Cover"	M33WEEKLY_SNOW
Product type identifier: MOD34_L3_MN	"MOD.AM1.vi_1m.L3"	M34L3_PROD_ID
NDVI	"NDVI_250_M"	M34NDVI
MVI	"MVI_250_M"	M34MVI
View zenith angles for NDVI	"VZ_NDVI"	M34NDVI_ZENITH_ANGLES
View zenith angles for MVI	"VZ_MVI"	M34MVI_ZENITH_ANGLES
Quality control for NDVI	"NDVI_250_M_QC"	M34NDVI_QUALITY
Quality control for MVI	"MVI_250_M_QC"	M34MVI_QUALITY
Product type identifier: MOD42_L3_WK_G	"MOD.AM1.V1.seaice_10dy.L3"	M42L3_PROD_ID
Weekly Sea Ice Cover	"Composite_Ice_Cover"	M42WEEKLY_SEA_ICE

(This page intentionally left blank.)

APPENDIX C: DESCRIPTIONS AND PURPOSES

Appendix C shows the descriptions and purposes for both the C and FORTRAN routines. For a description of the variables refer to Appendix D and for a description of the associated error messages see Appendix E.

C.1 Descriptions and Purposes of C Routines

`int` **`closeMODISfile`** `(MODFILE **`*`file`*`)`

closeMODISfile terminates the access of M-API routines to a MODIS HDF file opened using **openMODISfile**. Only pre-existing files should be closed by **closeMODISfile**. **completeMODISfile** should be used to end access to a new MODIS HDF file so that the file's header information can be completed. **closeMODISfile** may fail to close the file if an error occurs.

```
int completeMODISfile (MODFILE **file, PGSt_MET_all_handles mdHandles,
                       ECSattr_names_for_all_handles HDFattrNames, long
                       int NumHandles)
```

completeMODISfile terminates the access of M-API routines to a MODIS HDF file opened using **openMODISfile**. In addition to closing the file, the file's standard header information is inserted. A pre-existing MODIS-HDF file should be closed by **closeMODISfile** or some of its header information will be over-written. **completeMODISfile** may fail to close the file if an error occurs.

See Chapter 4.5, Accessing Metadata, for a complete list of metadata **completeMODISfile** writes to the MODIS-HDF file before closing it.

```
int createMODISarray (MODFILE *file, char *arrayname, char *groupname,
                      char *data_type,  long int rank,
                      long int dimsizes[])
```

createMODISarray creates an HDF SDS structure to store a new data array into a MODIS HDF file. It must be called before the data may be written to the file using **putMODISarray** or the attributes associated with the array may (optionally) be stored using **PMARIN** and **PMDMIN**.

The *groupname* string provides the facility to place the new array in an HDF 'Vgroup' data group. If a Vgroup with the name *groupname* does not exist, the array structure will not be created. The array may be placed in the file outside of any Vgroup by replacing *groupname* with NULL in C.

If an array with the name *arrayname* is written outside of a Vgroup, it must not already exist in the file. This is to prevent the confusion caused by multiple data objects with the same name. Arrays with the same name may be stored in the same file, however, if they are placed in separate Vgroups.

`int` **`createMODISgroup`** `(MODFILE *`*`file`*`, char *`*`groupname`*`, char *`*`classname`*`)`

createMODISgroup creates an HDF Vgroup structure in a MODIS HDF file to store table and array structures. It must be called before any of the data objects to be aggregated in it are created. The use of data groups is optional, but data objects stored in them are documented in the MODIS Product File Definitions in Appendix F. A data group with the name *groupname* must be unique in a file. This prevents confusion that is caused by multiple data groups with the same name.

```
int createMODIStable (MODFILE *file, char *tablename, char *classname,
                      char *groupname, char*fieldname, char*data_type)
```

createMODIStable creates an HDF Vdata structure in a MODIS HDF file to store a new data table. It must be called before the data may be written to the file using **putMODIStable**. The text headers for each field (column) and the data type stored in each field must be provided.

The *groupname* string provides the facility to place the new table in an HDF 'Vgroup' data group. If a Vgroup with the name *groupname* does not exist, the table structure will not be created. The table may be placed in the file outside of any Vgroup by setting *groupname* = NULL in C.

If a table with the name *tablename* is created outside of a Vgroup, it must not already exist in the file. This is to prevent the confusion caused by multiple data objects with the same name. Tables with the same name may be stored in the same file, however, if they are placed in separate Vgroups.

```
int endMODISobjaccess (MODFILE *file, char *name, char *group,
                       long int type)
```

endMODISobjaccess ends the access to an individual or a group of opened objects by deleting objects' DATAID structure from the ring super structure, releasing memory, and detaching (for Vdata) or end-accessing to (for SDS) the objects. This routine is called by **closeMODISfile** or **completeMODISfile**, which calls **closeMODISfile**, so that all opened objects will be closed automatically before the MODIS HDF file is closed. As long as an application program calls **closeMODISfile** or **completeMODISfile**, the application does not need to call this routine to close an object or a group of objects. However, if an application program determines an object will no longer be accessed and wish to end the access to the object for releasing computer resource, the application program can call this routines.

```
int getMODISardims (MODFILE *file, char *arrayname, char *groupname,
                    char *data_type, long int *rank, long int dimsizes[])
```

getMODISardims retrieves the essential characteristics of an HDF SDS array structure contained in a MODIS HDF file. This provides the information needed for properly reading data from the array structure using **getMODISarray**.

The *groupname* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrayname* if groupname = NULL in C.

Proper dimensioning of *dimsizes* to provide sufficient elements for the dimensions of the array structure may at first appear to require precognition. The easiest solution is to provide a generous (32 element) *dimsizes* array. Another approach is to use the *rank* variable as an input containing the number of elements in *dimsizes.* If *dimsizes* is inadequate for the multi-dimensional array structure in question, **getMODISardims** will fail gracefully but will return the rank of the array structure, allowing for the dimension information to be retrieved with a second call.

`int` **`getMODISarinfo`** `(MODFILE *`*`file`*`, char *`*`arrayname`*`, char *`*`groupname`*`, char *`*`attribute`*`, char *`*`data_type`*`, long int *`*`n_elements`*`, void *`*`value`*`)`

getMODISarinfo retrieves the value stored in an HDF local attribute associated with an array structure given the attribute name. If the attribute cannot be found, the routine will return MFAIL (-1) .

The routine will also fail if the provided *data_type* is found to be different than the metadata's data type or the *n_elements* is found to be too small to contain the number of metadata values. **getMODISarinfo** replaces this input information with the actual data type and number of elements contained in the metadata value (in the case of character data, it is the length of the string, including the '\0' terminator). The retrieved data type and attribute array size information may then be used to properly retrieve the array structure metadata with a second call to the routine. Since *data_type* and *n_elements* are used to output information, these arguments may not be pointers to constants. GMARIN behaves similarly, so the arguments *nelmnt* and *dtype* must not be FORTRAN parameters or constants either.

n_elements, the address of the number of elements in the provided output *value* array, is a required input if the metadata are to be retrieved. **getMODISarinfo** normally replaces this input with the actual array length required to hold this metadata. If the local attribute is not found or an HDF routine fails, however, **n_elements* is set to 0.

A variable of the proper data type and length should be passed for the *value* argument. The data type information required to properly use this routine may be found in Appendix F, MODIS Data Product File Definitions.

The *groupname* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrayname* if the argument*groupname* = NULL in C.

`int` **`getMODISarray`** `(MODFILE *`*`file`*`, char *`*`arrayname`*`, char *`*`groupname`*`, long int `*`start[]`*`, long int, `*`dimsizes[]`*`, void *`*`data`*`)`

getMODISarray returns a multi-dimensional array of data from an HDF SDS array structure contained in a MODIS HDF file. The data array must be of the same data type as data in the target array structure. In addition, the dimensions and array region requested from the array structure must be consistent with the structure's rank and dimensions. (The array structure's data type, rank, and dimensions may be retrieved using **getMODISardims**. If a **getMODISarray** error message occurs the data retrieval will not be performed. See Section 4.3, "Accessing Arrays" for additional information.

The *groupname* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrayname* if groupname = NULL in C.

```
int getMODISdiminfo (MODFILE *file, char *arrayname, char *groupname, long
                     int dimension, char *attribute, char *data_type,
                     long int *n_elements, void *value)
```

getMODISdiminfo retrieves the value stored in an HDF local attribute associated with an array structure's dimension given the attribute name. If the attribute cannot be found, the routine will return MFAIL (-1) .

The routine will also fail if the provided *data_type* is found to be different than the metadata's data type or the *n_elements* is found to be too small to contain the number of metadata values. getMODISdiminfo replaces this input information with the actual data type and number of elements contained in the metadata value (in the case of character data, it is the length of the string, including the '\0' terminator). The retrieved data type and attribute array size information may then be used to properly retrieve the array structure metadata with a second call to the routine. Since *data_type* and *n_elements* are used to output information, these arguments may not be pointers to constants. GMDMIN behaves similarly, so the arguments *nelmnt* and *dtype* must not be FORTRAN parameters or constants either.

n_elements, the address of the number of elements in the provided output *value* array, is a required input if the metadata are to be retrieved. **getMODISdiminfo** normally replaces this input with the actual array length required to hold this metadata. If the local attribute is not found or an HDF routine fails, however, **n_elements* is set to 0.

A variable of the proper data type should be passed for the *value* argument. The data type information required to properly use either routine may be found in Appendix F, MODIS Data Product File Definitions.

The *groupname* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrayname* if the argument*groupname* = NULL in C.

```
int getMODISdimname (MODFILE *file, char *arrayname, char *groupname, long
                     int dimension, char *dimname)
```

getMODISdimname retrieves the name of an HDF dimension associated with an array structure given the array's name and the dimension's number. If the dimension name cannot be found, the routine will return MFAIL (-1). This routine does not retrieve a " long_name" dimension attribute. **getMODISdiminfo** can retrieve such a dimension label (if it exists), however.

The *groupname* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively, the entire file will be searched for an array structure named *arrayname* if the argument *groupname* = NULL in C or *grpnm* is a blank string (' ') in FORTRAN.

```
int getMODISECSinfo (MODFILE *file,  char *PVLAttrName, char *parmName,
                     char *data_type, long int *n_elements,  void *value)
```

getMODISECSinfo is part of a larger software system called the MODIS Applications Programming Interface (API) Utility, abbreviated M-API. The M-API Utility consists of subroutines which allow MODIS Science Team-supplied software to read and write data and metadata from/to HDF files. The functionality of the M-API is defined in the MODIS Application Program Interface (API) Specification.

In HDF-EOS, parameters are collected together to form a text block using PVL. Then the text block is stored in HDF as a single attribute. **getMODISECSinfo** retrieve the value of a parameter from the PVL text block.

In order to obtain value of a parameter inside a PVL text block, the function reads the PVL text block specified by *PVLAttrName* from the MODIS file, creates the internal ODL tree structure from the PVL text block, and search the tree structure to retrieve the value of a parameter. The tree structure is then saved internally for consecutive searches in the same PVL text block for code efficiency. If multiple parameters will be retrieved from the same PVL block, just set *PVLAttrName* to the HDF PVL attribute name in the first call and set to NULL in C and ' ' in FORTRAN in the consecutive calls. If the next call is to retrieve the value of a parameter in a different PVL text block, set the PVLAttrName to the new PVL attribute name. The saved old tree structure will be deleted automatically and a new ODL tree will be created and saved. If you will no longer call getMODISECSinfo in your program and want to release the memory occupied by the saved tree, just set both *PVLAttrName* and *parmName* to NULL in C.

```
int getMODISfields (MODFILE *file,  char *tablename,  char *groupname, long
                    int *stringlen, long int *recno, long int *fieldno,
                    char *fieldname, char *data_type, char *classname)
```

getMODISfields retrieves the essential characteristics of an HDF Vdata table structure contained in a MODIS-HDF file. This provides the information needed for properly reading data from the table structure using **getMODIStable** or to write to it using **putMODIStable**. If any of the output parameters are set to NULL, then that data are not retrieved. An error (MFAIL) will be returned if 1) The output strings are not long enough to contain the data type or field name strings for all the Vdata's fields, 2) an unknown (e.g., not supported by the MODIS API) number type is encountered or 3) an HDF routine FAILs. The data type string (if requested) will be returned truncated to the point where the fault occurred.

stringlen, the address of the length of the *data_type* and *fieldname* output strings, is a required input if either of these strings is to be retrieved. getMODISfields normally replaces this input with the actual array length required to hold the larger ot the two output strings. If an unknown data type or an HDF routine fails, however, **stringlen* is set to 0.

The *groupname* string provides the facility to select a table structure existing in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for a table structure named *tablename* if *groupname* = NULL in C.

`int` **`getMODISfileinfo`** `(MODFILE *`*`file`*`, char *`*`attribute,`* `char *`*`data_type`*`, long int *`*`n_elements, void *value`*`)`

getMODISfileinfo retrieves the value associated with an attribute = value metadata pair given the attribute name. If the attribute cannot be found, the routine will return -1 and the passed variable unchanged.

The routine will also fail if the provided *data_type* is found to be different than the metadata's data type or the *n_elements* is found to be too small to contain the metadata's value. **getMODISfileinfo** replaces this input information with the actual data type and number of elements contained in the metadata value (in the case of character data, it is the length of the string). These metadata metadata may be used to properly retrieve the metadata value with a second call to the routine.

A variable of the proper data type should be passed for the *value* parameter. The data type information required to properly use either routine may be found in Appendix B, M-API-Supplied Constants, and Appendix F, MODIS Data Product File Definitions. Appendix B has a listing for each M-API provided metadata attribute that includes the data type, the format, and/or specific values associated with it.

`int` **`getMODIStable`** `(MODFILE *`*`file`*`, char *`*`tablename,`* `char *`*`groupname`*`, char*`*`fieldname`*`, long int `*`start`*`, long int `*`recno`*`, long int *`*`buffsize,`* `unsigned char *`*`data`*`)`

getMODIStable retrieves one or more fields of data from one or more records in an HDF Vdata table structure contained in a MODIS-HDF file. The data are placed in the *data* buffer in consecutive records and in the order that the input *fieldnames* are listed. The length of this buffer must be able to contain all the fields requested times the number of records requested. If the *buffsize* input indicates that it is too small to contain the actual quantity of data requested, **getMODIStable** will fail, but it will return the actual *buffsize* required. The output *data* buffer must be at least this size. See Section 4.4, "Accessing Tables" for additional information.

The *groupname* string provides the facility to select a table structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for a table structure named *tablename* if groupname = NULL in C.

`int` **`putMODISarinfo`** `(MODFILE *file, char *arrayname, char *groupname, char *attribute, char *data_type, long int n_elements, void *value)`

putMODISarinfo attachs a local metadata attribute/value pair to a MODIS array. **putMODISarinfo** stores an attribute = value(s) metadata pair to the indicated array. If the attribute already exists, the value(s) will be updated.

`int` **`putMODISarray`** `(MODFILE *`*`file`*`, char *`*`arrayname,`* `char *`*`groupname`*`, long int `*`start[],`* `long int, `*`dimsizes[]`*`, void*`*`data`*`)`

putMODISarray places a multi-dimensional array of data into an HDF SDS array structure previously created using **createMODISarray**. The data in the array must be of the data type the target array structure was created for. In addition, the dimensions and placement of the input array in the array structure must be consistent with the structure's rank and dimensions. If a **putMODISarray** error message occurs, the data insertion will not be performed. See Section 4.3, "Accessing Arrays" for additional information. This routine may be called multiple times to fill the array structure. Data previously in the array structure may be overwritten.

The *groupname* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. The entire file will be searched for an array structure named *arrayname* if groupname = NULL in C.

```
int putMODISdiminfo (MODFILE *file, char *arrayname, char *groupname, long
                     int dimension, char *attribute, char *data_type, long
                     int n_elements, void *value)
```

putMODISdiminfo attachs a local attribute/value pair to a specific dimension of a MODIS array. **putMODISdiminfo** stores an attribute = value(s) attribute pair to the indicated dimension of a MODIS array. If the attribute already exists, the value(s) will be updated.

The *groupname* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrayname* if groupname = NULL in C and a blank string (" ") in FORTRAN.

```
int putMODISdimname (MODFILE *file, char *arrayname, char *groupname, long
                     int dimension, char *dimname)
```

putMODISdimname associates an HDF dimension name with a specified SDS array structure dimension. The SDS array must be created (using **createMODISarray**) before it is possible to name any of its dimensions. This routine does not create a "long_name" dimension attribute. **putMODISdiminfo** can produce such a dimension label, however. **putMODISdimname** does more than apply an appellation to a dimension. An HDF dimension name is an independent data object. It may be shared by several array structure dimensions, but they all must be of the same size. Any dimension attribute that is associated with any one of these dimensions is immediately associated with all the dimensions having that name. Likewise, updating a dimension attribute for one dimension updates it for all dimensions having the same name (they could only have one "long name" dimension shared between them). Naming an SDS dimension will also cause any dimension attributes currently associated with that dimension to be lost. Therefore it is most practical to name an array's dimensions, if necessary, immediately after the array structure's creation and before creating dimension attributes for it. The *groupname* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. alternatively, the entire file will be searched for an array structure named *arrayname* if the argument *groupname* = NULL in C or *grpnm* is a blank string (" ") in FORTRAN.

```
int putMODISfileinfo (MODFILE *file, char *attribute, char *data_type,
                      long int n_elements, void * value)
```

putMODISfileinfo stores an attribute = value metadata pair to the indicated MODIS HDF file. If the attribute already exists, the value will be updated.

File attributes should be limited to M-API provided attribute macros. (See Section 5, M-API-Supplied Constants and Naming Conventions.) The data type should also be limited to the type associated with the MODIS file attribute, and the value itself restricted to that data type and the format and/or specific values associated with the attribute.

```
int putMODIStable (MODFILE *file, char *tablename, char *groupname, long
                   int start, long int recno, unsigned char *data)
```

putMODIStable places one or more data records into an HDF Vdata table structure previously created using **createMODIStable**. The data to be inserted into the table must be placed into a data array. The length of this array must be an integral number of the table structure's record length. The various data that make up a record should be inserted into the buffer in the same order as the field headers were ordered in the **createMODIStable** call. See Section 4.4, "Accessing Tables" for additional information. This routine may be called multiple times to fill the table structure. Data previously in the table structure may be overwritten.

The *groupname* string provides the facility to select a table structure placed in a particular HDF 'Vgroup' data group. The entire file will be searched for a table structure named *tablename* if *groupname* = NULL in.

int **substrMODISECSinfo** (*char *char_value*, long int *n_elements,* long int *n_strings,* char *substr[])

ECS metadata values may be integer, floating point, or character string values or arrays of values. Some may be multiple strings. The routine **getMODISECSinfo** retrieves such strings into a one-dimension character array with the individual strings separated by nulls ('\0'). **substrMODISECSinfo** breaks this 'packed' character array into its constituent substrings. **substrMODISECSinfo** sets the pointers in a provided output array to the beginning of each substring in the *char_value* array.

int32 **searchMODISgroup** (MODFILE *file, char *groupname, char *classname, char *objectname, char *objectclass, int32 objecttype)

searchMODISgroup searches an HDF Vgroup structure in a MODIS HDF file to find if an HDF object is in the Vgroup. Both the group and the object are specified by their name and class name. However, the classname is an optional feature. If class names are set to NULL, only name comparison is performed. Because SDS (array) has no class name, the objectclass for an SDS is always ignored. If the specified object exists, the function will return the reference id for Vdata and Vgroup, and index for SDS. If the object does not exist, the function will return NO_OBJECT, which is defined in mapic.h as -2.

The *groupname* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrayname* if groupname = NULL in C and a blank string (" ") in FORTRAN.

long int **MODISsizeof** (char *data_type)

The M-API uses a set of standard strings to describe the data types in stored in array and table structures. These strings are returned, for example, by the routine **getMODISardims** to describe the data type of the target array structure. **MODISsizeof** returns the number of bytes required to store a data type given this data type string. The input string may be a series of comma-delimited data type strings, in which case the total number of bytes to store the record described by the string is returned.

MODFILE * **openMODISfile** (char *filename, char *access)

openMODISfile opens an HDF file and creates the HDF structures to support the M-API routines access to it. openMODISfile must be called to produce the MODFILE structure before any of these routine can access it. Note that setting the file access to "w" creates a file and will overwrite a pre-existing one. Will close the file and return null outputs if an error occurs.

C.2 Descriptions and Purposes of FORTRAN Routines

INTEGER FUNCTION **CLMFIL** (*modfil*)

CLMFIL terminates the access of M-API routines to a MODIS HDF file opened using **OPMFIL**. Only pre-existing files should be closed by closeMODISfile. **CPMFIL** should be used to end access to a new MODIS HDF file so that the file's header information can be completed. **CLMFIL** may fail to close the file if an error occurs.

INTEGER FUNCTION **CPMFIL** *(modfil, mdhandle, hdfattrnms, numhands)*

CPMFIL terminates the access of M-API routines to a MODIS HDF file created using **OPMFIL**. In addition to closing the file, the MODIS file's standard header information is inserted. A pre-existing MODIS HDF file should be closed by **CLMFIL** or some of its header information will be overwritten. **CLMFIL** may fail to close the file if an error occurs.

See Section 4.5, Accessing Metadata, for a complete list of metadata **CPMFIL** writes to the MODIS HDF file before closing it.

INTEGER FUNCTION **CRMAR** (*modfil, arrnm, grpnm, dtype, rank, dims*)

CRMAR creates an HDF SDS structure to store a new data array into a MODIS HDF file. It must be called before the data may be written to the file using **PMAR** or the attributes associated with the array may (optionally) be stored using **PMARIN** and **PMDMIN**.

The *grpnm* string provides the facility to place the new array in an HDF 'Vgroup' data group. If a Vgroup with the name *groupname* does not exist, the array structure will not be created. The array may be placed in the file outside of any Vgroup by replacing *grpnm* = a blank string (' ') in FORTRAN.

If an array with the name *arrnm* is written outside of a Vgroup, it must not already exist in the file. This is to prevent the confusion caused by multiple data objects with the same name. Arrays with the same name may be stored in the same file, however, if they are placed in separate Vgroups.

INTEGER FUNCTION **CRMGRP** (*modfil, grpnm, clsnm*)

CRMGRP is part of a larger software system called the MODIS Applications Programming Interface (API) Utility, abbreviated M-API. The M-API Utility consists of subroutines which allow MODIS Science Team-supplied software to read and write data and metadata from/to HDF files. The functionality of the M-API is defined in the M-API Specification.

CRMGRP creates an HDF Vgroup structure in a MODIS HDF file to store table and array structures. It must be called before any of the data objects to be aggregated in it are created. The use of data groups is optional, but data objects stored in them are documented in the MODIS Product File Definitions in Appendix F. A data group with the name *grpnm* must be unique in a file. This prevents confusion that is caused by multiple data groups with the same name.

INTEGER FUNCTION **CRMTBL** (*modfil, tblnm, clsnm, grpnm, fldnm, dtype*)

CRMTBL creates an HDF Vdata structure in a MODIS HDF file to store a new data table. It must be called before the data may be written to the file using **PMTBL**. The text headers for each field (column) and the data type stored in each field must be provided.

The *grpnm* string provides the facility to place the new table in an HDF 'Vgroup' data group. If a Vgroup with the name *grpnm* does not exist, the table structure will not be created. The table may be placed in the file outside of any Vgroup by setting *grpnm* = ' ' in FORTRAN.

If a table with the name *tblnm* is created outside of a Vgroup, it must not already exist in the file. This is to prevent the confusion caused by multiple data objects with the same name. Tables with the same name may be stored in the same file, however, if they are placed in separate Vgroups.

int **EMOBJ** (*modfil, name, group, type*)

EMOBJ ends the access to an individual or a group of opened objects by deleting objects' DATAID structure from the ring super structure, releasing memory, and detaching (for Vdata) or end-accessing to (for SDS) the objects. This routine is called by **CLMFIL** or **CPMFIL**, which calls **CLMFIL**, so that all opened objects will be closed automatically before the MODIS HDF file is closed. As long as an application program calls **CLMFIL** or **CPMFIL**. However, if an application program determines an object will no longer be accessed and wish to end the access to the object for releasing computer resource, the application program can call this routines.

INTEGER FUNCTION **GMAR** (*modfil, arrnm, grpnm, start, dims, data)*

GMAR returns a multi-dimensional array of data from an HDF SDS array structure contained in a MODIS HDF file. The data array must be of the same data type as data in the target array structure. In addition, the dimensions and array region requested from the array structure must be consistent with the structure's rank and dimensions. (The array structure's data type, rank, and dimensions may be retrieved using **GMARDM**). If a **GMAR** error message occurs the data retrieval will not be performed. See Section 4.3, "Accessing Arrays" for additional information.

The *grpnm* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrnm* if *grpnm* = a blank string (' ') in FORTRAN.

INTEGER FUNCTION **GMARDM** (*modfil, arrnm, grpnm, dtype, rank, dims)*

GMARDM retrieves the essential characteristics of an HDF SDS array structure contained in a MODIS HDF file. This provides the information needed for properly reading data from the array structure using **GMAR**.

The *grpnm* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrnm* if *grpnm* = a blank string (" ") in FORTRAN.

Proper dimensioning of *dims* to provide sufficient elements for the dimensions of the array structure may at first appear to require precognition. The easiest solution is to provide a generous (32 element) *dims* array. Another approach is to use the *rank* variable as an input containing the number of elements in *dims.* If *dims* is inadequate for the multi-dimensional array structure in question, **GMARDM** will fail gracefully but will return the rank of the array structure, allowing for the dimension information to be retrieved with a second call.

INTEGER FUNCTION **GMARIN** (*modfil,arrnm,grpnm,attrib, dtype,nelmnt,value*)

GMARIN retrieves the value stored in an HDF local attribute associated with an array structure given the attribute name. If the attribute cannot be found, the routine will return MFAIL (-1) .

The routine will also fail if the provided *dtype* is found to be different than the metadata's data type or the *nelmnt* is found to be too small to contain the number of metadata values. **GMARIN** replaces this input information with the actual data type and number of elements contained in the metadata value (in the case of character data, it is the length of the string, including the '\0' terminator). The retrieved data type and attribute array size information may then be used to properly retrieve the array structure metadata with a second call to the routine. Since *dtype* and *nelmnt* are used to output information, these arguments may not be pointers to constants.

nelmnt, the address of the number of elements in the provided output *value* array, is a required input if the metadata are to be retrieved. **GMARIN** normally replaces this input with the actual array length required to hold this metadata. If the local attribute is not found or an HDF routine fails, however, *nelmnt* is set to 0.

A variable of the proper data type and length should be passed for the *value* argument. The data type information required to properly use this routine may be found in Appendix F, MODIS Data Product File Definitions.

The *grpnm* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrnm* if the argument*grpnm* = *grpnm* is a blank string (" ") in FORTRAN.

INTEGER FUNCTION **GMDMIN** (*modfil, arrnm, grpnm, dim, attrib, dtype, nelmnt, value*)

GMDMIN retrieves the value stored in an HDF local attribute associated with an array structure's dimension given the attribute name. If the attribute cannot be found, the routine will return MFAIL (-1) .

The routine will also fail if the provided *dtype* is found to be different than the metadata's data type or the *nelmnt* is found to be too small to contain the number of metadata values. **getMODISdiminfo** replaces this input information with the actual data type and number of elements contained in the metadata value (in the case of character data, it is the length of the string, including the '\0' terminator). The retrieved data type and attribute array size information may then be used to properly retrieve the array structure metadata with a second call to the routine. Since *dtype* and *nelmnt* are used to output information, these arguments may not be pointers to constants.

nelmnt, the address of the number of elements in the provided output *value* array, is a required input if the metadata are to be retrieved. **GMDMIN** normally replaces this input with the actual array length required to hold this metadata. If the local attribute is not found or an HDF routine fails, however, *nelmnt* is set to 0.

A variable of the proper data type should be passed for the *value* argument. The data type information required to properly use either routine may be found Appendix F, MODIS Data Product File Definitions.

The *grpnm* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrnm* if the argument*grpnm* = a blank string (" ") in FORTRAN.

INTEGER FUNCTION **GMDNAM** (*modfil, arrnm, grpnm, dim, dname*)

GMDNAM retrieves the name of an HDF dimension associated with an array structure given the array's name and the dimensions number. If the dimension name cannot be found, the routine will return MFAIL (-1). This routine does not retrieve a " long_name" dimension attribute. **GMDMIN** can retrieve such a dimension label (if it exists), however. The *grpnm* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively, the entire file will be searched for an array structure named *arrnm* if the argument is a blank string (' ') in FORTRAN.

INTEGER FUNCTION **GMECIN** (*modfil, pvlname, pname, nms, dtype, pvalue*)

GMECIN is part of a larger software system called the MODIS Applications Programming Interface (API) Utility, abbreviated M-API. The M-API Utility consists of subroutines which allow MODIS Science Team-supplied software to read and write data and metadata from/to HDF files. The functionality of the M-API is defined in the M-API Specification.

In HDF-EOS, parameters are collected together to form a text block using PVL. Then the text block is stored in HDF as a single attribute. GMECIN retrieve the value of a parameter from the PVL text block.

In order to obtain value of a parameter inside a PVL text block, the function reads the PVL text block specified by *pvlname* from the MODIS file, creates the internal ODL tree structure from the PVL text block, and search the tree structure to retrieve the value of a parameter. The tree structure is then saved internally for consecutive searches in the same PVL text block for code efficiency. If multiple parameters will be retrieved from the same PVL block, just set pvlname to the HDF PVL attribute name in the first call and set to ' ' in the consecutive calls. If the next call is to retrieve the value of a parameter in a different PVL text block, set the pvlname to the new PVL attribute name. The saved old tree structure will be deleted automatically and a new ODL tree will be created and saved. If you will no longer call **GMECIN** in your program and want to release the memory occupied by the saved tree, just set both *pvlname* and *pname* to ' ' .

INTEGER FUNCTION **GMFIN** (*modfil, attrib, dtype, nelmnt, value*)

GMFIN retrieves the value associated with an attribute = value metadata pair given the attribute name. If the attribute cannot be found, the routine will return -1 and the passed variable unchanged.

The routine will also fail if the provided *dtype* is found to be different than the metadata's data type or the *nelmnt* is found to be too small to contain the metadata's value. **GMFIN** replaces this input information with the actual data type and number of elements contained in the metadata value (in the case of character data, it is the length of the string). These metadata metadata may be used to properly retrieve the metadata value with a second call to the routine.

A variable of the proper data type should be passed for the *value* parameter. The data type information required to properly use either routine may be found in Appendix B, M-API-Supplied Constants, and Appendix F, MODIS Data Product File Definitions. Appendix B has a listing for each M-API provided metadata attribute that includes the data type, the format, and/or specific values associated with it.

INTEGER FUNCTION **GMFLDS** (*modfil, tblnm, grpnm, strln, recno, fldno, fldnm, dtype, clsnm*)

GMFLDS retrieves the essential characteristics of an HDF Vdata table structure contained in a MODIS-HDF file. This provides the information needed for properly reading data from the table structure using **GMTBL** or to write to it using **PMTBL**. If any of the output parameters are set to NULL, then that data are not retrieved. An error (MFAIL) will be returned if:

1) The output strings are not long enough to contain the data type or field name strings for all the Vdata's fields,

2) an unknown (e.g., not supported by the MODIS API) number type is encountered or

3) an HDF routine FAILs. The data type string (if requested) will be returned truncated to the point where the fault occurred.

stringlen, the address of the length of the *dtype* and *fname* output strings, is a required input if either of these strings is to be retrieved. **GMFLDS** normally replaces this input with the actual array length required to hold the larger ot the two output strings. If an unknown data type or an HDF routine fails, however, **stringlen* is set to 0.

The *grpnm* string provides the facility to select a table structure existing in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for a table structure named *tblnm* if *grpnm* = a blank string (' ') in FORTRAN.

INTEGER FUNCTION **GMTBL** (*modfil, tblnm, grpnm, fldnm, start, recno, buffsz, data)*

GMTBL retrieves one or more fields of data from one or more records in an HDF Vdata table structure contained in a MODIS-HDF file. The data are placed in the *data* buffer in consecutive records and in the order that the input *fldnm* are listed. The length of this buffer must be able to contain all the fields requested times the number of records requested. If the *buffsz* input indicates that it is too small to contain the actual quantity of data requested, **GMTBL** will fail, but it will return the actual *buffsz* required. The output *data* buffer must be at least this size. See Section 4.4, "Accessing Tables" for additional information.

The *grpnm* string provides the facility to select a table structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for a table structure named *tblnm* if *grpnm* = a blank string (' ') in FORTRAN.

INTEGER FUNCTION **MSIZE** (*dtype*)

The M-API uses a set of standard strings to describe the data types in stored in array and table structures. These strings are returned, for example, by the routine **GMARDM** to describe the data type of the target array structure. **MSIZE** returns the number of bytes required to store a data type given this data type string. The input string may be a series of comma-delimited data type strings, in which case the total number of bytes to store the record described by the string is returned.

INTEGER FUNCTION **OPMFIL** (*fname*, *access*, *modfil*)

OPMFIL opens an HDF file and creates the HDF structures to support the M-API routines access to it. **OPMFIL** must be called to produce the FORTRAN modfil array before any of these routine can access it. Note that setting the file access to "w" creates a file and will overwrite a pre-existing one. **OPMFIL** will close the file and return null outputs if an error occurs.

INTEGER FUNCTION **PMAR** (*modfil, arrnm, grpnm, start, dims, data)*

PMAR places a multi-dimensional array of data into an HDF SDS array structure previously created using **CRMAR**. The data in the array must be of the data type the target array structure was created for. In addition, the dimensions and placement of the input array in the array structure must be consistent with the structure's rank and dimensions. If a **PMAR** error message occurs, the data insertion will not be performed. See Section 4.3, "Accessing Arrays" for additional information. This routine may be called multiple times to fill the array structure. Data previously in the array structure may be overwritten.

The *grpnm* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. The entire file will be searched for an array structure named *arrnm* if *grpnm* = a blank string (" ") in FORTRAN.

INTEGER FUNCTION **PMARIN** (*modfil, arrnm, grpnm, dtype, nelmnt, value)*

PMARIN stores an attribute = value metadata pair in an HDF local attribute associated with an array. The SDS array structure must be created (using **CRMAR**) prior to attaching a dimension attribute to it If the attribute already exists, the value(s) are updated.

The *grpnm e* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively, the entire file will be searched for an array structure named *arrnm* if the argument *grpnm* = NULL in C.

INTEGER FUNCTION **PMDMIN** (*modfil, arrnm, grpnm, dtype, nelmnt, value)*

PMDMIN stores an attribute = value metadata pair in an HDF local attribute associated with an array structure's dimension. The SDS array structure must be created (using **CRMAR**) prior to attaching a dimension attribute to it. If the attribute already exists, the value(s) are updated.

The *grpnm* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrnm* if the argument *grpnm e* = NULL in C.

INTEGER FUNCTION **PMDNAM**(*modfil, arrnm, grpnm, dim, dnm*)

PMDNAM associates an HDF dimension name with a specified SDS array structure dimension. The SDS array must be created (using **CRMAR**) before it is possible to name any of its dimensions. This routine does not create a "long_name" dimension attribute. **PMDNAM** can produce such a dimension label, however. **PMDNAM** does more than apply an appellation to a dimension. An HDF dimension name is an independent data object. It may be shared by several array structure dimensions, but they all must be of the same size. Any dimension attribute that is associated with any one of these dimensions is immediately associated with all the dimensions having that name. Likewise, updating a dimension attribute for one dimension updates it for all dimensions having the same name (they could only have one "long name" dimension shared between them). Naming an SDS dimension will also cause any dimension attributes currently associated with that dimension to be lost. Therefore it is most practical to name an array's dimensions, if necessary, immediately after the array structure's creation and before creating dimension attributes for it. The *grpnm* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively, the entire file will be searched for an array structure named *arrnm* if the argument *grpnm* = NULL in C or *grpnm* is a blank string (" ") in FORTRAN.

INTEGER FUNCTION **PMFIN** (*modfil, attrib, dtype, nelmnt, value*)

PMFIN stores an attribute = value metadata pair to the indicated MODIS HDF file. If the attribute already exists, the value will be updated.

File attributes should be limited to M-API provided attribute macros. (See Section 5, M-API-Supplied Constants and Naming Conventions.) The data type should also be limited to the type associated with the MODIS file attribute, and the value itself restricted to that data type and the format and/or specific values associated with the attribute.

INTEGER FUNCTION **PMTBL** (*modfil, tblnm, grpnm, start, recno, datasz, data)*

PMTBL places one or more data records into an HDF Vdata table structure previously created using **CRMTBL**. The data to be inserted into the table must be placed into a data array. The length of this array must be an integral number of the table structure's record length. The various data that make up a record should be inserted into the buffer in the same order as the field headers were ordered in the **CRMTBL** call. See Section 4.4, "Accessing Tables" for additional information. This routine may be called multiple times to fill the table structure. Data previously in the table structure may be overwritten.

The *grpnm* string provides the facility to select a table structure placed in a particular HDF 'Vgroup' data group. The entire file will be searched for a table structure named *tblnm* if *grpnm* = ' ' in FORTRAN.

INTEGER FUNCTION **SMECIN** (*cvalue*, *nelmnt*, *nstrs*, *substr*)

ECS metadata values may be integer, floating point, or character string values or arrays of values. Some may be multiple strings. The routine **GMECIN** retrieves such strings into a one-dimension character array with the individual strings separated by nulls ('\0'). SMECIN breaks this 'packed' character array into its constituent *substr*ings. **SMECIN** copies these *substr*ings into separate rows of a FORTRAN character string array.

INTEGER FUNCTION **SRMGRP** (modfil, grpnm, clsnm, objnm, objcls, objtyp)

SRMGRP searches an HDF Vgroup structure in a MODIS HDF file to find if an HDF object is in the Vgroup. Both the group and the object are specified by their name and class name. However, the classname is an optional feature. If class names are set to NULL, only name comparison is performed. Because SDS (array) has no class name, the objectclass for an SDS is always ignored. If the specified object exists, the function will return the reference id for Vdata and Vgroup, and index for SDS. If the object does not exist, the function will return NO_OBJECT. The NO_OBJECT is defined in mapic.inc as -2.

The *groupname* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrayname* if groupname = NULL in C and a blank string (" ") in FORTRAN.

(This page intentionally left blank.)

APPENDIX D: VARIABLES FOR ROUTINES

Table D-1. Variables for C Routines

Parameter	Data Type	Definition
access	char *	IN: Standard C access mode. One of: "r" Open for read only. "w" Create for read/write, over writes pre-existing files. "a" Open for read/write, creates a file that doesn't exist.
arrayname	char *	IN: ASCII string that will be the name of the array, up to 256 characters long. Array names cannot begin with a blank character and trailing blanks should be removed or else FORTRAN programs will have difficulty accessing them.
attribute	char *	IN: ASCII string name of the attribute. Provided macros for accepted MODIS HDF file attribute names are listed in Appendix B, M-API-Supplied Constants.
buffsize	long int *	IN/OUT: Address of the *data* buffer size on input, in bytes. The buffer must be at least this size. *buffsize* will normally return the number of bytes of data successfully retrieved. If the buffer is too small, however, the routine returns MFAIL and *buffsize* will contain the size a buffer must be to contain the output data. If a functional error occurs, it is set to 0 because making this output size determination will be unreliable.
char_value	char *	IN: Character string containing the 'packed' multiple substrings of ECS metadata retrieved with getMODISECSinfo. Do not deallocate *char_value* until *substr* array gets correct values.
classname	char *	IN: ASCII string that will be the class name of the table, up to 64 characters long. If set to NULL or an empty string, the table will have no class. OUT: ASCII string for the class name of the table structure. Provided array may be up to 64 bytes long.
data	void * and unsigned char *	IN/OUT: Address of the data buffer.

Parameter	Data Type	Definition
data_type	char *	IN/OUT: Address of the data type of the *value* output. The attribute's value will not be retrieved unless the input data type matches that of the attribute. NOTE: This argument must not be a the address of a constant string and should point to memory at least 8 bytes long. Permitted C data types: "int8" "uint8" "int16" "uint16" "int32" "uint32" "float32" "float64" "char *"
dimension	long int *	IN: The dimension number which the attribute is attached to (0-based). getMODISdiminfo associates the 0 dimension with the least rapidly varying array index of an HDF SDS array structure.
dimname	char *	IN: ASCII string for the dimension name. Provided array should be at least 256 bytes long.
dimsizes	long int *	IN: The size of the array being retrieved from the array structure. The *dimsize* array must have the same number of elements as the target array structure has dimensions and the product of the array dimensions must equal the number of elements in *data.* OUT: Array describing the size of each dimension of the target HDF array structure. The dimensions will not be provided unless dimsizes contains sufficient elements for the rank of the array.
ECSattr_names_for_all_handles	long int *	IN: A character array with size of [PGSd_MET_NUM_OF_GROUPS] [MAX_ECS_NAME_L], where PGSd_MET_NUM_OF_GROUPS] is 20 and MAX_ECS_NAME_L is 50. This array is typedef-ed as ECSattr_names_for all_handles. Each row in this array is a character string used as a global attribute name for storing an ECS PVL text block which has a handle in the corresponding row in mdHandles array. Each name, which is a string, should be less that MAX_ECS_NAME_L characters and occupies one row in the array. Specifies the number of actual handles contained in mdHandles. This may be set from 0 to PGSd_MET_NUM_OF_GROUPS.

Parameter	Data Type	Definition
fieldname	char *	IN: Array of comma-delimited ASCII string table headers. The headers should be in the same order that the data for each table row will subsequently be written in. Each field name must be less than 128 characters long and the Vdata table may contain up to 36 fields. OUT: Array of comma-delimited ASCII string table headers.
fieldno	long int *	OUT: Number of fields (columns) present in the table structure.
file	Modfile*	IN/OUT: Pointer to MODFILE structure address used to reference a file in all M-API routines. Set to NULL when the file is successfully closed.
filename	char *	IN: Path and filename for the file to be opened, up to 255 characters long.
group	char *	IN: The name of the group to which objects belongs. If *group* is set to NULL, all lone objects (objects belonging to no group) matched with *name* and *type* will be closed. If both *name* and *group* are NULL, all objects matched with *type* will be closed.
groupname	char *	IN: ASCII string name of the data group containing the target array structure. For 'GET' functions: If set to NULL the entire file will be searched for the array structure named *arrayname.* For 'PUT' functions: If set to NULL or an empty string, the table will not be placed in a data group.
HDFattrNames	Modfile*	IN: A character array with size of [PGSd_MET_NUM_OF_GROUPS] [MAX_ECS_NAME_L], where PGSd_MET_NUM_OF_GROUPS is 20 and MAX_ECS_NAME_L is 256. This array is typedef-ed as *ECSattr_names_for_all_handles*. Each row in this array is a character string used as a global attribute name for storing an ECS PVL text block which has a handle in the corresponding row in *mdHandles* array. Each name, which is a string, should be less than MAX_ECS_NAME_L characters and occupies one row in the array.
mdHandles	Modfile*	IN: A character array with size of [PGSd_MET_NUM_OF_GROUPS] [PGSd_MET_GROUP_NAME_L], where PGSd_MET_NUM_OF_GROUPS is 20 and PGSd_MET_GROUP_NAME_L is 50. This array is typedef-ed as **PGSt_MET_all_handles**. Each row in the array stores a handle to an internal ODL tree structure which will be written out as an ECS PVL attribute. Each handle, which is a string, should be less than 50 characters and occupy one row in the array. Therefore, the maximum number of handles should be 20.

Parameter	Data Type	Definition
n_elements	long int *	IN/OUT: Address of the number of memory elements as *data_type* available in the *value* array. The attribute's value will not be retrieved unless **n_elements* indicates that there is sufficient space available in *value*. getMODISECSinfo replaces this input with the number of elements required to contain the metadata. If the parameter cannot be found, **n_element* will be left unchanged, or set to 0 if a function error occurs. NOTE: This argument must not be the address of a constant.
n_elements (continued)	long int *	**SPECIAL CASE for multiple strings:** If there are multiple character strings for the parameters, strings will be packed together and returned in *value* . The separator between strings is '\0'. The low 16 bit of *n_elements* will return the total bytes in the values, including the '\0's between the strings and the '\0' at the end of last string. The part above the low 16 bits will return number of strings packed - 1. To obtain how many string retrieved, do the calculation: n_strings = *n_elemets/65536 + 1 n_bytes = *n_elements%65536 Therefore, if **n_elements* is less than 65536, there is only one strings in *value* and **n_elements* is the number of bytes (characters) in the string, including the last '\0'.
n_strings	long int *	IN/OUT: Address of the number of pointers available in the *substr* array. The *substr* pointers will not be set to the substrings in *char_value* unless there are sufficient pointers available in the pointer array. substrMODISECSinfo replaces this input with the number of substrings pointers have been set to in the *char_value* array. **n_strings* will be set to 0 if a function error occurs. This argument must not be the address of a constant.
name	char *	IN: The name of the object. If the name is set to NULL, all objects matched with *group* and object *type* will be closed.
NumHandles	long int *	IN: Specifies the number of actual handles contained in mdHandles. This may be set from 0 to PGSd_MET_NUM_OF_GROUPS.
objectclass	char *	IN: (Optional)ASCII string of the class name of the data object. Set to NULL for not comparing the object class
objectname	char *	IN: ASCII string of the object name to be searched.
objecttype	int32	IN: Type of the object; The valid objects are: DFTAG_NDG (for SDS) DFTAG_VH (for Vdata, or attribute if the object class is set to Attr0.0) DFTAG_VG (for Vgroup).

Parameter	Data Type	Definition
parmName	char *	IN: ASCII string name of a parameter whose value will be retrieved. Set both PVLAttrName and parmName to NULL in C will release the memory occupied by the internal ODL tree. The parmName could parameter name only or combination of name and class represented as "name.class".
PGSt_MET_all_handles	char *	IN: A character array with size of [pGSd_MET_NUM_OF_GROUPS] [PGS_MET_GROUP_NAME_L], where PGSd_MET_NUM_OF_GROUPS is 20 and PGSd_MET_GROUP_NAME_L is 50. This array is typedef-ed as PGSt_MET_all_handles. Each row in the array stores a handles to an internal ODL tree structure which will be written out a an ECS PVL attribute. Each handles, which is a string, should be less than 50 characters and occupy, one row in the array. Therefore, the maximum number of handles should be 20.
PVLAttrName	char *	IN: ASCII string name of the HDF attribute which contains the PVL text block. Set PVLAttrName to NULL in C while parmName is not NULL in C or not ' ' in FORTRAN will result in searching the last PVL text block for the value of *parmName* parameter.
rank	long int *	IN/OUT: The number of elements in the array *dimsizes* on input. This is replaced with the number of dimensions in the target HDF array structure for output. It is set to 0 if a functional error occurs. No dimensional information will be provided if rank = NULL.
recno	long int *	IN: Number of records being inserted into the table structure. OUT: Number of records(rows) present in the table structure. NOTE: The product of *recno* and the table structure's record length must have the same length as the buffer addressed by *data*
reproc_status	char *	IN: Intent to reprocess the data.
start	long int *	IN: Zero-based record location to begin placing reading the data into the table structure. NOTE: The *start* array must have the same number of elements as the target array structure has dimensions. The *start* location must be contiguous to the location of records already in the table. For placing If *start* = -1 data records will be appended to the end of the table structure.
stringlen	long int *	IN/OUT: Input of the minimum length of *fieldname* and *data_type* arrays. Returns the minimum array length actually required to hold the longer of the two strings. It is set to 0 if a functional error occurs.
substr	char *	OUT: Array of poiners to the constituent substrings contained in the *char_value* array.

Parameter	Data Type	Definition	
tablename	char *	IN:	ASCII string that will be the name of the table, up to 64 characters long. Table names should not include trailing blanks or else FORTRAN programs will have difficulty accessing them.
temporal_coverage	char *	IN:	Description observation period in ECS metadata syntax.
type	long int	IN:	The object type: MODIS_ARRAY (for SDS, the numerical value is 720, the same value as DFTAG_NDG), MODIS_TABLE(for Vdata, the numerical value is 1962, the same value as DFTAG_VH), or MODIS_ALL_TYPES (the numerical values is 0). If *type* is 720, only SDS objects will be closed. If *type* is 1962, only Vdata will be closed. If *type* is MODIS_ALL_TYPES, both Vdata and SDS objects specified by *name* and *group* will be closed. Therefore, to close all opened objects, set both *name* and *group* to NULL and set *type* to 0.
value	void	IN:	Address of the data to store in the in the attribute. If the attribute already exists, the value will be updated. Values should conform to the data types, formats and/or those values enumerated for the attribute in Appendix B, M-API-Supplied Constants.
		OUT:	Buffer for the value. User should allocate enough memory for this buffer. If there are multiple data values in character type, the value will be placed consecutively. If the data value type is "char *", string will be separated by '\0'.

Table D-2. Variables for FORTRAN Routines

Parameter	Data Type	Definition
access	Character*(*)	IN: Standard C access mode. One of: 'r' Open for read only. 'w' Create for read/write. 'a' Open for read/write (append.).
arrnm	Character*(*)	IN: ASCII string name of the target HDF array structure, up to 128 characters long. Array names cannot begin with a blank character.
attrib	Character*(*)	IN: ASCII string name of the attribute. Provided macros for accepted MODIS HDF file attribute names are listed in Appendix B, M-API-Supplied Constants.
buffsize	Integer	IN/OUT: The *data* buffer size on input, in bytes. The buffer must be at least this size. *buffsize* will normally return the number of bytes of data successfully retrieved. If the buffer is too small, however, the routine returns MFAIL and *buffsize* will contain the size a buffer must be to contain the output data requested. If a functional error occurs, it is set to 0 because making this output size determination will be unreliable.
clsnm	Character*(*)	IN: ASCII string that will be the class name of the table, up to 64 characters long. If set to a blank string, the table will have no class. OUT: ASCII string for the class name of the table structure. Provided array should be at least (64) bytes long.
cvalue	Character*(*)	IN: Character string containing the 'packed' multiple substrings of ECS metadata retrieved with GMECIN.
data	<any>	IN/OUT: Multi-dimensional data array. NOTE:
dim	Integer	IN: The dimension number which the attribute is attached to (0-based). GMDMIN associates the 0 dimension with the most rapidly varying array index of an HDF SDS array structure.
dims	Integer	IN: The size of the array being inserted into the array structure. The *dims* array must have the same number of elements as the target array structure has dimensions and the product of the array dimensions must equal the number of elements in *data.* OUT: Array describing the size of each dimension of the target HDF array structure. The dimensions will not be provided unless dims contains sufficient elements for the rank of the array. (HDF 3.3r4 SDS's may contain up to 32 dimensions.)
dname	Character*(*)	IN: ASCII string for the dimension name. Provided array should be at least 256 bytes long.
dnm	Character*(*)	IN: ASCII string name to give to the dimension.

Parameter	Data Type	Definition
dtype	Character*(*)	IN/OUT: Data type of the *value* output. The attribute's value will not be retrieved unless the input data type matches that of the attribute. GMARIN replaces with the data type of the retrieved metadata. NOTE: This argument must not be a parameter or constant. The memory size of *dtype* should be at least 13 characters long. Permitted FORTRAN data types: 'INTEGER*1' 'UINTEGER*1' 'INTEGER*2' 'UINTEGER*2' 'INTEGER*4' 'UINTEGER*4' 'REAL*4' 'REAL*8' 'CHARACTER*(*)'
fldnm	Character*(*)	IN: Array of comma-delimited ASCII string table headers. The headers should be in the same order that the data for each table row will subsequently be written in. Each field name must be less than 128 characters long and the Vdata table may contain up to 36 fields. OUT: Array of comma-delimited ASCII string table headers.
fldno	Integer	OUT: Number of fields (columns) present in the table structure.
fname	Character*(*)	IN/OUT: Number of elements availabel in the *value* array. Output replaces with the number of elements required to contain the metadata.
grpnm	Character*(*)	IN: ASCII string name of the data group containing the target array structure. OUT: ASCII string name of the data group to place the new array in. For 'GET' functions: If *grpnm* = ' ' the entire file will be searched for the array structure named *arrnm/tblnm.* For 'PUT' functions: If set to " "(blank) the array will not be placed in a data group.
hdfatrnms	Character*255(*)	IN: A character array with size of [PGSd_MET_NUM_OF_GROUPS] [MAX_ECS_NAME_L-1], where PGSd_MET_NUM_OF_GROUPS is 20 and MAX_ECS_NAME_L is 256. Each string in this array is a character string used as a global attribute name for storing an ECS PVL text block which has a handle in the corresponding row in mdHandles array. Each name, which is a string, should be less that MAX_ECS_NAME_L characters and occupies one row in the array.

Parameter	Data Type	Definition
mdhandle	Character*45(*)	IN: An array of character strings. The memory size of the array is [PGSd_MET_NUM_OF_GROUPS] [PGS_MET_GROUP_NAME_L-1], where PGSd_MET_NUM_OF_GROUPS is 20 and PGSd_MET_GROUP_NAME_L is 50. This array is typedef-ed as PGSt_MET_all_handles. Each row in the array stores a handles to an internal ODL tree structure which will be written out a an ECS PVL attribute. Each handles, which is a string, should be less than 50 characters and occupy, one row in the array. Therefore, the maximum number of handles should be 20.
modfil	Integer	IN: Array that is used to reference a MODIS HDF file in all other M-API routines. OUT: Array that is used to reference the file in all other M-API routines. The array will return all zeroes if an error occurs.
nelmnt	Integer	IN: The composite output dimensions, from GMECIN, containing (in the case of character string metadata the total length (in bytes) of the string in *cvalue* in its lower two bytes and the number of substrings packed into *cvalue* less one in the upper two bytes. The calculations: n_strings = n_elements/65536 + 1 n_bytes = n_elements%65536 provide the number of substrings and the total length, respectively, of the data in *cvalue*. When there is only one string in *cvalue*, *nelmnt* will be less than 65536 and there is no need to use SMECIN. OUT: Number of elements availabel in the *value* array. The attribute's value will not be retrieved unless *nelmnt* indicates that there is sufficient space in *value*. Output replaces with the number of elements required to contain the metadata. If a function error occurs, however, *nelmnt* is set to 0. This argument must not be a parameter or constant.
nms	Character*(*)	IN/OUT: The number of memory elements as *dtype* available in the *value* array. The attribute's value will not be retrieved unless *nms* indicates that there is sufficient space available in *value*. GMECIN replaces this input with the number of elements required to contain the metadata. If the parameter cannot be found, **nms* will be left unchanged, or set to 0 if a function error occurs. This argument must be a variable.

Parameter	Data Type	Definition
nms (continued)	Character*(*)	**SPECIAL CASE for multiple strings:** If there are multiple character strings for the parameters, strings will be packed together and returned in *value* . The separator between strings is '\0' (numerical value 0). The low 16 bit of *nms* will return the total bytes in *value*, including the '\0's. The part above the low 16 bits will return (number of strings packed - 1). To obtain how many string retrieved, do the calculation: n_strings = nms/65536 + 1 n_bytes = MOD(nms, 65536) Therefore, if *nms* is less than 65536, there is only one strings in *value* and *nms* is the number of bytes (characters) in the string.
nstrs	Integer	IN/OUT: Number of elements available in the *substr* array. The *substr* will not be set to the substrings in *cvalue* unless there are sufficient elements available in the *substr* array. SMECIN replaces this input with the number of substrings already set in the *cvalue* array. *nstrs* will be set to 0 if a function error occurs.
numhands	Integer	IN: Specifies the number of actual handles contained in mdHandles. This may be set from 0 to PGSd_MET_NUM_OF_GROUPS.
objcls	Character*(*)	IN: (Optional)ASCII string of the class name of the data object. Set to NULL for not comparing the object class.
objnm	Character*(*)	IN: ASCII string of the object name to be searched.
objtyp	Integer	IN: type of the object; The valid objects are: DFTAG_NDG, DFTAG_VH, DFTAG_VG.
pname	Character*(*)	IN: ASCII string name of a parameter whose value will be retrieved. Set both *pvlname* and *pname* to ' ' will release the memory occupied by the internal ODL tree. The pname could parameter name only or combination of name and class represented as "name.class".
pvlname	Character*(*)	IN: ASCII string name of the HDF attribute which contains the PVL text block. Set *pvlname* to ' ' while pname is not equal to ' ' will result in searching the last PVL text block for the value of *pname* parameter.
rank	Integer	IN/OUT: The number of elements in the array *dimsizes* on input. This is replaced with the number of dimensions in the target HDF array structure for output. It is set to 0 if a functional error occurs.
recno	Integer	IN: Number of records being inserted into the table structure. The product of *recno* and the table structure's record length must have the same length as the buffer addressed by *data.* OUT: Number of records(rows) present in the table structure.
reproc	Character*(*)	IN: Intent to reprocess the data.

Parameter	Data Type	Definition
start	Integer	IN: Zero-based record location to begin placing the data into the table structure. The *start* location must be contiguous to the location of records already in the table. If *start* = -1 data records will be appended to the end of the table structure. The *start* array must have the same number of elements as the target array has dimensions.
stringlen	Integer	IN/OUT: Minimum length of *fldnm* and *dtype* arrays. Returns the minimum array length actually required to hold the longer of the two strings. It is set to 0 if a functional error occurs.
substr	Character*(*)	OUT: Array of substrings obtained from the *cvalue* array.
tblnm	Character*(*)	IN: ASCII string that will be the name of the table, up to 64 characters long.
tcov	Character*(*)	IN: Description observation period in ECS metadata syntax.
value	<valid type>	IN: Data to store in the in the attribute. If the attribute already exists, the value will be updated. Values should conform to the data types, formats and/or those values enumerated for the attribute in Appendix B, M-API-Supplied Constants. OUT: Value associated with the attribute.

(This page intentionally left blank.)

APPENDIX E: ERROR MESSAGES FOR ROUTINES

Table E-1. Error Messages

Routine	Error Message	Description
closeMODISfile (CLMFIL)	closeMODISfile cannot close a null file.	
	closeMODISfile detected FAIL from HDF function Sdend. Unable to close *filename*.	File could not be closed because access identifiers to data objects are still attached to the file. Changes to the file may be lost.
	WARNING: closeMODISfile closed new file *filename* without complete header information.	The file has been successfully closed, but completeMODISfile should be used instead so that the required file header information will be included.
completeMODISfile (CPMFIL)	completeMODISfile unable to continue with empty input.	
	closeMODISfile detected FAIL from HDF function Hclose. Unable to close file.	
	closeMODISfile detected FAIL from HDF function Sdend. Unable to close file.	
	completeMODISfile detected FAIL from HDF procedure Hclose. Unable to close file.	File could not be closed because access identifiers to data objects are still attached to the file. Changes to the file may be lost.
	WARNING: completeMODISfile revised header data of pre-existing filename file.	The file has been successfully closed, but closeMODISfile should be used instead to prevent modification to the MODIS HDF file's metadata.
	WARNING: completeMODISfile unable to revise header data of filename file open for read-only.	The file has been successfully closed and was accessed only for reading.

Routine	Error Message	Description
createMODISarray (CRMAR)	createMODISarray unable to make a new *arrayname* array with a NULL file MODFILE structure.	
	createMODISarray unable to make a new array without an array name input.	
	createMODISarray unable to make a new *arrayname* array without array dimension input.	
	createMODISarray unable to make a new *arrayname* array without array data type input.	
	createMODISarray unable to make a new *arrayname* array in file opened for read only.	
	createMODISarray found *arrayname* array already exists.	
	createMODISarray found *arrayname* array already exists in data group *"groupname"*.	
	createMODISarray unable to find data group *groupname* to place new *arrayname* array in.	
	createMODISarray unable to create *arrayname* array of data type *data_type.*	
	createMODISarray unable to create *arrayname* array with *rank* dimensions.	
	createMODISarray detected FAIL from HDF procedure SDcreate attempting to create *arrayname* array.	
	createMODISarray detected FAIL from HDF procedure Sdend access while attempting to create *arrayname* array.	
	createMODISarray detected FAIL from HDF procedure Vattach attempting to create *arrayname* array.	
	createMODISarray detected FAIL from HDF procedure Vaddtagref attempting to create *arrayname* array.	

Routine	Error Message	Description
createMODISarray (CRMAR) (continued)	createMODISarray detected FAIL from HDF procedure Vdetach attempting to create *arrayname* array.	
createMODIStable (CRMTBL)	createMODIStable unable to make a new table without a table name input.	
	createMODIStable unable to make a new *tablename* table with a NULL file MODFILE structure.	
	createMODIStable unable to make a new *tablename* table without field names input.	
	createMODIStable unable to make a new *tablename* table without field data types input.	
	createMODIStable unable to make a new *tablename* table in file opened for read only.	
	createMODIStable found the *tablename* table already exists.	
	createMODISarray found *arrayname* array already exists in data group "*groupname*".	
	createMODISarray unable to find data group *groupname* to place new *arrayname* array in.	
	createMODIStable unable to create *tablename* table with # byte records.	Vdata table records are limited to 32K each.
	createMODIStable unable to create *tablename* table with *data_type* data types.	
	createMODIStable unable to allocate memory for *fieldname* temporary buffer used to create the *tablename* table.	
	createMODIStable unable to allocate memory for *data_type* temporary buffer used to create *tablename* table.	
	createMODIStable found the *tablename* table to have no fields in the fieldname string *fieldname.*	

Routine	Error Message	Description
createMODIStable (CRMTBL) (continued)	createMODIStable unable to support the creation of # fields in the field name string "fieldname" for the "tablename" table.	Vdata table records are limited to fields
	createMODIStable found the *tablename* table to have # data types in the data type string *data_type* instead of #.	One data type must be supplied for each field in the Vdata table.
	createMODIStable detected FAIL from HDF procedure VSattach attempting to create the *tablename* table.	
	createMODIStable detected fail from HDF procedure VSfdefine for *field* and *data_type* of the *tablename* table.	
	createMODIStable detected FAIL from HDF procedure VSsetfields creating the *tablename* table.	
	createMODIStable unable to allocate memory for dummy field buffer used to create the *tablename* table.	
	createMODIStable detected FAIL from HDF procedure VSwrite creating the *tablename* table.	
	createMODIStable detected FAIL from HDF procedure Vattach attempting to create the *tablename* table.	
	createMODIStable detected FAIL from HDF procedure Vaddtagref attempting to create the *tablename* table.	
	createMODIStable detected FAIL from HDF procedure Vdetach attempting to create the *tablename* table.	
	nccrmtble failed at data_type conversion.	
	nccrmtble out of memory	
endMODISobjaccess (EMOBJ)	endMODISobjaccess unable to close object *name* with an invalid MODIS file structure input.	
	endMODISobjaccess unable to close objects with an invalid MODIS object type: *type*.	

Routine	Error Message	Description
endMODISobjaccess (EMOBJ) (continued)	endMODISobjaccess detected FAIL from HDF procedure SDendaccess while closing access to array *did->name*. endMODISobjaccess detected FAIL from HDF procedure VSdetach while closing access to table *did->name*.	
getMODISdimname (GMDNAM)	getMODISdimname unable to read the name of a dimension without the name of array it is associated with. getMODISdimname unable to read the name of a dimension in the *arrayname* array with an invalid MODIS file structure input. getMODISdimname unable to read the name of a *arrayname* array's dimension name without an output character string. getMODISdimname detected MFAIL from M-API internal function getMODISarrayid while attempting to obtain the name of dimension *dimension* in the *arrayname* array. getMODISdimname unable to read the dimension name of the non-existent dimension *dimension* of the *arrayname* array. getMODISdimname detected FAIL from HDF procedure SDgetdimid attempting to read the name of an *arrayname* array's dimension. getMODISdimname detected FAIL from HDF procedure SDsetdimname attempting to read the name of an *arrayname* array's dimension.	
getMODISardims (GMARDM)	getMODISardims unable to access the *arrayname* array with a NULL file MODFILE structure. getMODISardims unable to access an array without an array name input. getMODISardims unable to return the *arrayname* array's dimensions without a dimsizes array.	

Routine	Error Message	Description
getMODISardims (GMARDM) (continued)	getMODISardims cannot find the *arrayname* array.	
	getMODISardims cannot find the *arrayname* array in the *groupname* data group.	
	getMODISardims unable to find the *groupname* data group containing the *arrayname* array.	
	getMODISardims cannot get an sds_id for the *arrayname* array.	
	getMODISardims detected FAIL from HDF procedure SDgetinfo attempting to access the *arrayname* array.	
	getMODISardims detected FAIL from HDF procedure SDendaccess attempting to detach from the *arrayname* array.	The output from getMODISardims may not be valid if SDendaccess fails.
	*rank (if provided) is set to 0 if any of the errors associated with these messages occurs.	
	getMODISardims unable to return the *arrayname* array's *sds_rank* dimension sizes in a *rank* element dimsizes array.	getMODISardims will not attempt to write to the dimsizes output array, but it will return the rank of the target HDF array structure. The dimsizes array needs to have at least this many elements.
getMODISarinfo (GMARIN)	getMODISarinfo unable continue with empty n_elements.	
	getMODISarinfo unable to access an array attribute without an attribute name input.	
	getMODISarinfo unable to access the *attribute* attribute without the name of the array it is associated with.	No *arrayname* attribute was provided.
	getMODISarinfo cannot find array "*arrayname*".	

Routine	Error Message	Description
getMODISarinfo (GMARIN) (continued)	getMODISarinfo unable to find the *groupname* data group containing the *arrayname* array.	This may be preceeded by one of the following three messages:
	searchMODISgroup fails to search object *objectname* in group *groupname* because Vattach fails.	
	searchMODISgroup unable to find the specified Vgroup group *groupname*.	
	searchMODISgroup fails to obtain *objectname*'s tag and reference number.	
	getMODISarinfo cannot find the *arrayname* array in the *groupname* data group.	The Vdata table could not be found in the specified Vgroup data group.
	getMODISarinfo detected FAIL retrieving the data type string for the *attribute* attribute using DFNT_to_datatype.	M-API currently does not recogniize the HDF number types 3 (unsigned char), 7 (float128), 27 (unsigned int64), 28 (int128), 30 (unsigned int128), 42 (char16), 43 (unsigned char 16), or any greater than 512 (machine specific, custom, or little endian storage formats).
	getMODISarinfo cannot find local array attribute *attribute*.	
	getMODISarinfo detected FAIL from HDF procedure SDselect attempting to read the *attribute* attribute.	
	getMODISarinfo detected FAIL from HDF procedure SDattrinfo attempting to read the *attribute* attribute.	
	getMODISarinfo detected FAIL from HDF procedure SDreadattr attempting to read the *attribute* attribute.	
	getMODISarinfo unable to read local array attribute without output buffer for *attribute*.	
	getMODISarinfo detected FAIL from HDF procedure SDendaccess attempting to read the *attribute* attribute.	**n_elements* is set to 0 if any of the errors associated with the messages above occur.

Routine	Error Message	Description
getMODISarinfo (GMARIN) (continued)	WARNING: Vgroup groupname contains non-existing SDS object with reference id *ref_id*.	Information about an SDS array structure that doesn't really exist has been found in the Vgroup data group being accessed. While this will not directly prevent reading the specified local array attribute, it does identify a probable defect in the HDF file.
getMODISarray (GMAR)	getMODISarray unable to read from the *arrayname* array with a NULL file MODFILE structure.	
	getMODISarray unable to read from an array without an array name input.	
	getMODISarray unable to read from the *arrayname* array without array dimension input.	
	getMODISarray unable to read from the *arrayname* array without a data buffer.	
	getMODISarray cannot find the *arrayname* array.	
	getMODISarray cannot find the *arrayname* array in the *groupname* data group.	
	getMODISarray unable to find the *groupname* data group containing the *arrayname* array.	
	getMODISarray unable to read data from invalid array structure locations in the *arrayname* array.	This error message may be preceeded by one of the following two messages:
	SDS_footprintOK detected FAIL from HDF procedure Sdgetinfo.	
	Unable to access data at invalid array structure locations *"start[0] ... start[r]".*	
	getMODISarray detected FAIL from HDF procedure SDselect while attempting to read from the *arrayname* array.	

Routine	Error Message	Description
getMODISarray (GMAR) (continued)	getMODISarray detected FAIL from HDF procedure SDgetinfo while attempting to read from the *arrayname* array.	
	getMODISarray detected FAIL from HDF procedure SDwritedata while attempting to read from the *arrayname* array.	
	getMODISarray detected FAIL from HDF procedure SDendaccess while attempting to read from the *arrayname* array.	
getMODISdiminfo (GMDMIN)	getMODISdiminfo unable continue with empty n_elements.	
	getMODISdiminfo unable to access an array attribute without an attribute name input.	
	getMODISdiminfo unable to access the *attribute* attribute without the name of the array it is associated with.	No *arrayname* attribute was provided.
	getMODISdiminfo cannot find array "*arrayname*".	
	getMODISdiminfo unable to find the *groupname* data group containing the *arrayname* array.	This may be preceeded by one of the following three messages:
	searchMODISgroup fails to search object *objectname* in group *groupname* because Vattach fails.	
	searchMODISgroup unable to find the specified Vgroup group *groupname*.	
	searchMODISgroup fails to obtain *objectname*'s tag and reference number.	
	getMODISdiminfo cannot find the *arrayname* array in the *groupname* data group.	The Vdata table could not be found in the specified Vgroup data group.

Routine	Error Message	Description
getMODISdiminfo (GMDMIN) (continued)	getMODISdiminfo detected FAIL retrieving the data type string for the *attribute* attribute using DFNT_to_datatype.	M-API currently does not recogniize the HDF number types 3 (unsigned char), 7 (float128), 27 (unsigned int64), 28 (int128), 30 (unsigned int128), 42 (char16), 43 (unsigned char 16), or any greater than 512 (machine specific, custom, or little endian storage formats).
	getMODISdiminfo cannot find local array dimension attribute *attribute.*	
	getMODISdiminfo detected FAIL from HDF procedure SDselect attempting to read the *attribute* attribute	
	getMODISdiminfo detected FAIL from HDF procedure SDgetinfo attempting to read the *attribute* attribute.	
	getMODISdiminfo detected FAIL from HDF procedure SDattrinfo attempting to read the *attribute* attribute.	
	getMODISdiminfo unable to retrieve an *attribute* attribute for dimension *dimension*. The *arrayname* array has *rank* dimensions.	
	getMODISdiminfo detected FAIL from HDF procedure SDgetdimid attempting to read the *attribute* attribute.	
	getMODISdiminfo detected FAIL from HDF procedure SDreadattr attempting to read the *attribute* attribute.	
	getMODISdiminfo unable to read local array attribute without output buffer for *attribute*.	
	getMODISdiminfo detected FAIL from HDF procedure SDendaccess attempting to read the *attribute* attribute.	**n_elements* is set to 0 if any of the errors associated with the messages above occur.

Routine	Error Message	Description
getMODISdiminfo (GMDMIN) (continued)	WARNING: Vgroup groupname contains non-existing SDS object with reference id *ref_id*.	Information about an SDS array structure that doesn't really exist has been found in the Vgroup data group being accessed. While this will not directly prevent reading the specified local array attribute, it does identify a probable defect in the HDF file.
getMODISECSinfo (GMECIN)	getMODISECSinfo can not continue without the n_elements input. getMODISECSinfo unable to access an ECS metadata without the parameter name input. getMODISECSinfo unable to access the *parmName* metadata without the name of the global attribute it is stored within. getMODISECSinfo unable to access the *parmName* metadata from ECS global attribute *PVLAttrName* without the data type input. getMODISECSinfo detected fails in procedure MPVL2ODL while attempting to retrieve parameter *parmName* from ECS global attribute *PVLAttrName*. getMODISECSinfo can not find the *parmName* metadata. getMODISECSinfo found the value for parameter *parmName* is undefined. getMODISECSinfo unable to access the *parmName* metadata without the output data buffer. getMODISECSinfo found unknown ODL value type *valueNode->item.type* for parameter *parmName*.	
getMODISfields (GMFLDS)	getMODISfields unable to access the *tablename* table with a NULL file MODFILE structure. getMODISfields unable to access a table without a table name input.	

Routine	Error Message	Description
getMODISfields (GMFLDS) (continued)	getMODISfields cannot find *tablename* table.	
	getMODISfields unable to find the *groupname* data group containing the *tablename* table.	This may be preceeded by one of the following two messages:
	searchMODISgroup fails to search object *objectname* in group *groupname* because Vattach fails.	
	searchMODISgroup unable to find the specified Vgroup group *groupname*.	
	getMODISfields cannot find the *tablename* table in the *groupname* data group.	This may be preceeded by the following message:
	searchMODISgroup fails to obtain *objectname*'s tag and reference number.	The Vdata table could not be found in the specified Vgroup data group.
	getMODISfields detected FAIL from HDF procedure VSattach attempting to access the *tablename* table.	
	getMODISfields detected FAIL from HDF procedure VSgetfields.	A problem occurred with retrieving information about the number and names of the table's fields.
	getMODISfields detected FAIL retrieving the data type string for the *tablename* table using Vfdatatypes.	This error message may be preceeded by one of the following two messages:
	VFdatatypes detected FAIL from HDF routine Vfnfields.	
	VFdatatypes detected unrecognized HDF number type.	
	M-API currently does not recognize number types 3 (unsigned char), 7 (float128), 27 (unsigned int64), 28 (int128), 30 (unsigned int128), 42 (char16), 43 (unsigned char 16), or any greater than 512 (machine specific, custom, or little endian storage formats).	
	getMODISfields detected FAIL from HDF procedure VSinquire.	A problem occurred with retrieving information about the number of records in the table.

Routine	Error Message	Description
getMODISfields (GMFLDS) (continued)	getMODISfields detected FAIL from HDF procedure VSinquire.	A problem occurred with retrieving information about the number of records in the table. **stringlen* is set to 0 if any of the errors associated with the messages above occur.
	getMODISfields unable to fit *tablename* table's <string length> byte field names string into output string of unknown length.	The length of the output string *fieldname* was not provided in the parameter *stringlen*.
	getMODISfields unable to fit the *tablename* table's <string length> byte field names into **stringlen* byte output string.	*stringlen* will return the array length required to hold the table's field names.
	getMODISfields unable to fit *tablename* table's data types string into output string of unknown length.	The length of the output string *data_type* was not provided in the parameter *stringlen*.
	getMODISfields unable to fit the *tablename* table's <string length> byte data types into **stringlen* byte output string. VFdatatypes unable to fit data types into output string.	This error message will be preceeded by: **stringlen* will return the array length required to hold the table's data type string. If both the field names and the data types were requested, the larger of the two array lengths is returned.
	ncgmflds out of memory.	
getMODISfileinfo (GMFIN)	getMODISfileinfo detected FAIL from HDF procedure SDattrinfo. getMODISfileinfo detected FAIL from HDF procedure SDreadattr.	
getMODIStable (GMTBL)	getMODIStable unable continue without buffer size information. getMODIStable unable to read from the *tablename* table with a NULL file MODFILE structure. getMODIStable unable to read from a table without a table name input.	A location for *buffsize* information was not provided.

Routine	Error Message	Description
getMODIStable (GMTBL) (continued)	getMODIStable unable to read from the *tablename* table without a data buffer.	
	getMODIStable cannot find *tablename* table.	
	getMODIStable unable to find the *groupname* data group containing the *tablename* table.	This may be preceeded by one of the following two messages:
	searchMODISgroup fails to search object *objectname* in group *groupname* because Vattach fails.	
	searchMODISgroup unable to find the specified Vgroup group *groupname*.	
	getMODIStable cannot find the *tablename* table in the *groupname* data group.	This may be preceeded by the following message:
	searchMODISgroup fails to obtain *objectname*'s tag and reference number.	The Vdata table could not be found in the specified Vgroup data group.
	getMODIStable detected FAIL from HDF procedure VSattach attempting to access the *tablename* table.	
	getMODIStable unable to read data from the *tablename* table from invalid table structure record *start.*	
	getMODIStable unable to read data from the *tablename* table from invalid table structure locations.	Either access to some records or one or more fields requested do not exist in the table.
	getMODIStable detected FAIL from HDF procedure VSsetfields attempting to read *tablename* table.	
	getMODIStable detected FAIL from HDF procedure VSsizeof attempting to read *tablename* table.	
	getMODIStable detected FAIL from HDF procedure VSseek attempting to read *tablename* table.	
	getMODIStable detected FAIL from HDF procedure VSread attempting to read *tablename* table.	**buffsize* is set to 0 if any of the errors associated with the messages above occurs.

Routine	Error Message	Description
getMODIStable (GMTBL) (continued)	getMODIStable detected FAIL from HDF procedure VSinquire.	Should this error occur, getMODIStable will still return MAPIOK (because the data were successfully retrieved) and **buffsize* is set correctly.
	getMODIStable unable to fit <output size> bytes of *tablename* table's data into a *buffsize* byte output buffer.	getMODIStable will not attempt to write to the *data* output buffer, but it will return the buffer length (in bytes) required to hold the requested records from the table.
	WARNING: Vgroup groupname contains non-exist Vdata object with reference id *ref_id*.	Information about a Vdata table that doesn't really exist has been found in the Vgroup data group being accessed. While this will not directly prevent reading the specified Vdata table, it does identify a probable defect in the HDF file.
	WARNING: getMODIStable retrieved dummy record from empty table *tablename.*	The record retrieved from the table does not contain geophysical data. getMODIStable returns MAPIOK (0), however. This situation can only occur if NO geophysical data were written into the table or the single record in the Vdata was not written using M-API.
openMODISfile (OPMFIL)	openMODISfile unable to access a file without a filename input.	
	openMODISfile unable to open file *filename* without access mode input.	
	openMODISfile unable to allocate memory for a MODIS file structure for file *filename.*	
	openMODISfile unable to recognize access type *access* to open file *filename*.	

Routine	Error Message	Description
openMODISfile (OPMFIL) (continued)	openMODISfile unable to find file *filename.*	
	openMODISfile detected FAIL from HDF procedure SDstart opening file *filename*.	May be unable to open the HDF file because it is write-protected.
	openMODISfile detected NULL from HDF function SDIhandle_from_id accessing file *filename*.	
	openMODISfile unable to allocate memory for the MODIS filename *filename*.	
putMODISarinfo (PMARIN)	putMODISarinfo unable to write an array attribute without an attribute name input.	
	putMODISarinfo unable to write the *attribute* array attribute without data type information.	
	putMODISarinfo unable to write the *attribute* array attribute without the value buffer.	
	putMODISarinfo unable to write the *attribute* array attribute without the name of the array it is associated with.	No *arrayname* argument was provided.
	putMODISarinfo unable to write *n_elements attribute* array attribute values.	
	putMODISarinfo unable to write the *attribute* array attribute in a file opened for read only.	
	putMODISarinfo cannot find array *arrayname*.	
	putMODISarinfo unable to find the *groupname* data group containing the *arrayname* array.	This may be preceeded by one of the following three messages:
	searchMODISgroup fails to search object *objectname* in group *groupname* because Vattach fails.	
	searchMODISgroup unable to find the specified Vgroup group *groupname*.	
	searchMODISgroup fails to obtain *objectname*'s tag and reference number.	

Routine	Error Message	Description
putMODISarinfo (PMARIN) (continued)	putMODISarinfo cannot find the *arrayname* array in the *groupname* data group.	The SDS array structure could not be found in the specified Vgroup data group.
	putMODISarinfo unable to write the *attribute* array attribute with a *size* byte value.	Each HDF attribute is limited to 32K of memory.
	putMODISarinfo unable to write the *attribute* array attribute of data type *data_type*.	
	putMODISarinfo detected FAIL from HDF procedure SDselect attempting to write the *attribute* array attribute.	
	putMODISarinfo detected FAIL from HDF procedure SDsetattr attempting to write the *attribute* array attribute.	
	putMODISarinfo detected FAIL from HDF procedure SDendaccess attempting to write the *attribute* array attribute	
	WARNING: Vgroup groupname contains non-existing SDS object with reference id *ref_id*.	Information about an SDS array structure that doesn't really exist has been found in the Vgroup data group being accessed. While this will not directly prevent writing the specified local array attribute, it does identify a probable defect in the HDF file.
putMODISarray (PMAR)	putMODISarray unable to write to the *arrayname* array with a NULL file MODFILE structure.	
	putMODISarray unable to write to an array without an array name input.	
	putMODISarray unable to write to the *arrayname* array without array dimension input.	
	putMODISarray unable to write to the *arrayname* array without a data buffer.	

Routine	Error Message	Description
putMODISarray (PMAR) (continued)	putMODISarray unable to write to the *arrayname* array in file opened for read only.	
	putMODISarray cannot find the *arrayname* array.	
	putMODISarray cannot find the *arrayname* array in the *groupname* data group.	
	putMODISarray unable to find the *groupname* data group containing the *arrayname* array.	
	putMODISarray unable to write data to invalid array structure locations in the *arrayname* array.	This error message may be preceeded by one of the following two messages:
	SDS_footprintOK detected FAIL from HDF procedure SDgetinfo.	
	Unable to access data at invalid array structure locations *"start[0] ... start[r]".*	
	putMODISarray detected FAIL from HDF procedure SDselect while attempting to write to the *arrayname* array.	
	putMODISarray detected FAIL from HDF procedure SDgetinfo while attempting to write to the *arrayname* array.	
	putMODISarray detected FAIL from HDF procedure SDwritedata while attempting to write to the *arrayname* array.	
	putMODISarray detected FAIL from HDF procedure SDendaccess while attempting to write to the *arrayname* array.	
putMODISdiminfo (PMDMIN)	putMODISdiminfo unable to write an dimension attribute without an attribute name input.	
	putMODISdiminfo unable to write the *attribute* dimension attribute without data type information.	
	putMODISdiminfo unable to write the *attribute* dimension attribute without the value buffer.	

Routine	Error Message	Description
putMODISdiminfo (PMDMIN) (continued)	putMODISdiminfo unable to write the *attribute* dimension attribute without the name of the array it is associated with.	No *arrayname* argument was provided.
	putMODISdiminfo unable to write *n_elements attribute* dimension attribute values.	
	putMODISdiminfo unable to write the *attribute* dimension attribute in a file opened for read only.	
	putMODISdiminfo cannot find array *arrayname*.	
	putMODISdiminfo unable to find the *groupname* data group containing the *arrayname* array.	This may be preceeded by one of the following three messages:
	searchMODISgroup fails to search object *objectname* in group *groupname* because Vattach fails.	
	searchMODISgroup unable to find the specified Vgroup group *groupname*.	
	searchMODISgroup fails to obtain *objectname*'s tag and reference number.	
	putMODISdiminfo cannot find the *arrayname* array in the *groupname* data group.	The SDS array structure could not be found in the specified Vgroup data group.
	putMODISdiminfo unable to write the *attribute* dimension attribute with a *size* byte value.	Each HDF attribute is limited to 32K of memory.
	putMODISdiminfo unable to write the *attribute* dimension attribute of data type *data_type*.	
	putMODISdiminfo detected FAIL from HDF procedure SDselect attempting to write the *attribute* dimension attribute.	
	putMODISdiminfo detected FAIL from HDF procedure SDgetinfo attempting to write the *attribute* dimension attribute.	
	putMODISdiminfo detected FAIL from HDF procedure SDselect attempting to write the *attribute* dimension attribute.	

Routine	Error Message	Description
putMODISdiminfo (PMDMIN) (continued)	putMODISdiminfo unable to write the *attribute* attribute to non-existing dimension *dimension* of the *arrayname* array. putMODISdiminfo detected FAIL from HDF procedure SDgetdimid attempting to write the *attribute* dimension attribute. putMODISdiminfo detected FAIL from HDF procedure SDsetattr attempting to write the *attribute* dimension attribute. putMODISdiminfo detected FAIL from HDF procedure SDendaccess attempting to write the *attribute* dimension attribute. WARNING: Vgroup groupname contains non-existing SDS object with reference id *ref_id*.	Information about an SDS array structure that doesn't really exist has been found in the Vgroup data group being accessed. While this will not directly prevent writing the specified local dimension attribute, it does identify a probable defect in the HDF file.
putMODISdimname (PMDNAM)	putMODISdimname unable to name a dimension without a dimension name input. putMODISdimname unable to name a dimension *dimname* without the name of array the dimension is associated with. putMODISdimname unable to name a dimension *dimname* in the *arrayname* array with an invalid MODIS file structure input. putMODISdimname unable to name a dimension *dimname* in a file opened for read only. putMODISdimname detected MFAIL from M-API internal function getMODISarrayid while attempting to name a dimension *dimname* in the *arrayname* array.	

Routine	Error Message	Description
putMODISdimname (PMDNAM) (continued)	putMODISdimname unable to name a non-existing dimension *dimension dimname*. putMODISdimname detected FAIL from HDF procedure SDgetdimid attempting to name a dimension *dimname*. putMODISdimname detected FAIL from HDF procedure SDdiminfo attempting to name a dimension *dimname*. WARNING: putMODISdimname detected *nattrs* attributes currently attached to the dimension. Naming the *dimension* dimension of the *arrayname* array *dimname* will lose those attributes. putMODISdimname detected FAIL from HDF procedure SDsetdimname attempting to name a dimension *dimname*.	
putMODISfileinfo (PMFIN)	putMODISfileinfo unable continue with empty input. putMODISfileinfo unable to store *n_elements attribute* global attribute values. putMODISfileinfo unable to write metadata in file opened for read only. putMODISfileinfo unable to identify data type "*data*_type". putMODISfileinfo unable to write *attribute* metadata with a *size* byte value. putMODISfileinfo detected FAIL from HDF procedure SDsetattr.	
putMODIStable (PMTBL)	putMODIStable unable to write to the *tablename* table with a NULL file MODFILE structure. putMODIStable unable to write to a table without an table name input. putMODIStable unable to write to the *tablename* table without table dimension input.	

Routine	Error Message	Description
putMODIStable (PMTBL) (continued)	putMODIStable unable to write to the *tablename* table without a data buffer.	
	putMODIStable unable to write to the *tablename* table in file opened for read only.	
	putMODIStable cannot find the *tablename* table.	
	putMODIStable cannot find the *tablename* table in the *groupname* data group.	
	putMODIStable unable to find the *groupname* data group containing the *tablename* table.	
	putMODIStable detected FAIL from HDF procedure Vattach while attempting to write to the *tablename* table.	
	putMODIStable detected FAIL from HDF procedure VSattach while attempting to write to the *tablename* table.	
	putMODIStable detected FAIL from HDF procedure VSinquire while attempting to write to the *tablename* table.	
	putMODIStable unable to place *datasize* bytes of data into *recno record size* byte records in *tablename* table.	
	putMODIStable unable to write data to table *tablename* to invalid table structure record *start.*	The *start* location must be contiguous to the location of records already in the table.
	putMODIStable detected FAIL from HDF procedure VSseek while attempting to write to the *tablename* table.	
	putMODIStable detected FAIL from HDF procedure VSwrite while attempting to write to the *tablename* table.	

Routine	Error Message	Description
putMODIStable (PMTBL) (continued)	putMODIStable detected FAIL from M-API procedure set_Vhasdata while attempting to write to the *tablename* table. Sometimes it is necessary to read from the table structure before writing to it. The following two error messages may occur only in these circumstances: putMODIStable memory allocation failure while attempting to write to the *tablename* table. putMODIStable detected FAIL from HDF procedure VSread while attempting to write to the *tablename* table. putMODIStable memory allocation failure while attempting to write to the *tablename* table. putMODIStable detected FAIL from HDF procedure VSread while attempting to write to the *tablename* table.	The first record has successfully been written to the table, however M-API was unable to write an associated attribute into the file. This will cause a subsequent write to the table appending additional records to inadvertantly overwrite this first one.
substrMODISECSinfo (SMECIN)	substrMODISECSinfo unable to continue without char_value input. substrMODISECSinfo unable to continue without n_strings input. substrMODISECSinfo unable to continue without substr pointer array. substrMODISECSinfo unable to continue with invalid *n_elements* n_elements. substrMODISECSinfo unable to fit *loc_n_strings* substrings into **n_strings* pointers *substr* array. substrMODISECSinfo detected MFAIL from MAPI procedure parse_string attempting to parse the *char_value* char_value.	

(This page intentionally left blank.)

APPENDIX F: EXAMPLES OF MODIS DATA PRODUCT FILE DEFINITION

Version: V1 Revision: 7 Date: 24 April 1996

Product Identifier: MOD.AM1.V1.snow.L2, MOD10_L2

Description: Snow cover data product generated for granules of MODIS L1B data.

Product consists of an HDF file of a two SDSs and metadata. One SDS is the snow data as coded integers of algorithm results. The other SDS is the mandatory 1km Data Samples Per Line.

Contents:

SDS: Snow_Cover 1km Data Samples Per Line

Global Metadata:

Name:	Type:	Num_Val:	Source:	Value:
CoreMetadata.0	HDF-STRING	1	SDPtk	Variable
This string will contain the following PVL fields:				
ANCILLARYINPUTPOINTER	ECS-STRING	FREE_RANGE	Code	Variable
AUTOMATICQUALITYFLAG	ECS-STRING	1	Code	Variable
EASTBOUNDINGCOORDINATE	ECS-DOUBLE	1	Input	Variable
EXCLUSIONGRINGFLAG	ECS-STRING	1	Input	"N"
GRANULENUMBER	ECS-INTEGER	1	Input	Variable
GRANULEPOINTER	ECS-STRING	1	Code	Variable
GRINGPOINTLATITUDE	ECS-DOUBLE	4	Input	Variable
GRINGPOINTLONGITUDE	ECS-DOUBLE	4	Input	Variable
GRINGPOINTSEQUENCENO	ECS-INTEGER	4	Input	Variable
INPUTPOINTER	ECS-STRING	FREE_RANGE	Code	Variable
LONGNAME	ECS-STRING	1	Code	"Snow Cover"
MODISPRODUCTFILENAME	ECS-STRING	1	Code	Variable
NORTHBOUNDINGCOORDINATE	ECS-DOUBLE	1	Input	Variable
OPERATIONALQUALITYFLAG	ECS-STRING	1	Code	"not being investigated"
OPERATIONMODE	ECS-STRING	1	Input	Variable
ORBITNUMBER	ECS-INTEGER	1	Input	Variable
PROCESSINGDATETIME	ECS-STRING	1	Code	Variable
PROCESSINGHISTORYPOINTER	ECS-STRING	1	Code	Variable
QAPERCENTINTERPOLATEDDATA	ECS-INTEGER	1	Code	Variable
QAPERCENTMISSINGDATA	ECS-INTEGER	1	Code	Variable
QAPERCENTOUTOFBOUNDSDATA	ECS-INTEGER	1	Code	Variable
QUALITYFLAGEXPLANATION	ECS-STRING	FREE_RANGE	Code	Variable
RANGEBEGINNINGDATETIME	ECS-STRING	1	Input	Variable
RANGEENDINGDATETIME	ECS-STRING	1	Input	Variable
REPROCESSINGACTUAL	ECS-STRING	1	Code	"processed once"
REPROCESSINGPLANNED	ECS-STRING	1	Code	"no further update is anticipated"
SCIENCEQUALITYFLAG	ECS-STRING	1	Code	"not being investigated"
SHORTNAME	ECS-STRING	1	Code	"MOD10_L2"
SIZEMBECSDATAGRANULE	ECS-INTEGER	1	Code	Variable
SOUTHBOUNDINGCOORDINATE	ECS-DOUBLE	1	Input	Variable
SPSOPARAMETERS	ECS-STRING	2	Code	"3020"
WESTBOUNDINGCOORDINATE	ECS-DOUBLE	1	Input	Variable

Figure F-1. MODIS Data Product File Definition Examples

Global Metadata:

Name:	Type:	Num_val:	Source:	Value:
ProductMetadata.0	HDF-STRING	1	SDPtk	Variable
This string will contain the following PVL fields:				
ALGORITHMPACKAGEACCEPTANCEDATE	ECS-STRING	1	Code	"1997-01-01"
ALGORITHMPACKAGEMATURITYCODE	ECS-STRING	1	Code	"pre-launch"
ALGORITHMPACKAGENAME	ECS-STRING	1	Code	"MOD10V1"
ALGORITHMPACKAGEVERSION	ECS-STRING	1	Code	"version 1"
INSTRUMENTNAME	ECS-STRING	1	Code	"Moderate-Resolution Imaging SpectroRadiometer"
PLATFORMSHORTNAME	ECS-STRING	1	Code	"EOS AM1"
PROCESSINGCENTER	ECS-STRING	1	Code	"GSFC"
Number of Instrument Scans	HDF-uint16	1	Input	Variable

SDS Definition

SDS Name: Snow_Cover

Description: Snow cover extent as identified by the algorithm for every pixel in the granule. Coded values are; 200=snow, 100= not snow, 1=no decision, 0=fill, i.e. no input data.

Data conversions:

file data = (value * scale_factor) + add_offset value = (file data -add_offset)/scale_factor

Number Type: "unit8"

Rank: 2

Dimension sizes: (Data Lines, Maximum Number of Data Samples Per Line)

Dimension names: Dimension0: Data Lines

Dimension1: Maximum Number of Data Samples Per Line

SDS Metadata:

Name:	Type:	Num_val:	Source:	Value:
add_offset	HDF-float64	1	code	-127.0
add_offset_err	HDF-float64	1	code	0.0
calibrated_nt	HDF-int32	1	code	24
long_name	HDF-STRING	1	code	"Snow_covered_land"
scale_factor	HDF-float64	1	code	1.0
scale_factor_err	HDF-float64	1	code	0.0
units	HDF-STRING	1	code	"N/A"
valid_range	HDF-uint8	2	code	1,254
_FillValue	HDF-unit8	1	code	0
Data Lines	HDF-uint16	1	code	Variable
Maximum Number of Data Samples Per Line	HDF-uint16	1	code	Variable
Nadir Data Resolution	HDF-STRING	1	code	"500 m"
Number_of_pixels_processed	int32	1	code	Variable
Total_snow_pixels	int32	1	code	Variable
Total_not_snow_pixels	int32	1	code	Variable

Figure F-1. MODIS Data Product File Definition Examples (Continued)

SDS Metadata:

Name:	Type:	Num_val:	Source:	Value:
Area_snow	int32	1	code	Variable (km^2)
Area_not_snow	int32	1	code	Variable (km^2)
Percentage_snow	int16	1	code	Variable (%)
Percentage_not_snow	int16	1	code	Variable (%)
Above_range_NDSI	int32	1	code	Variable
Below_range_NDSI	int32	1	code	Variable
Division_by_zero	int32	1	code	Variable
Out_of_range_input	int32	1	code	Variable
No_decision	int32	1	code	Variable
Solar_zenith>=85	int32	1	code	Variable
Cloud_obscured	int32	1	code	Variable
QA_overall	string	1	code	Variable

SDS Definition
SDS Name: nLw_443
Description: 1km Data Samples Per 1km Data Line in this MODIS Granule

Number Type:	"int32"
Rank:	1
Dimension sizes:	(1km data lines contained in this granule)
Dimension names:	Dimension0: 1km Data Lines contained in this granule

SDS Metadata:

Name:	Type:	Num_val:	Source:	Value:
long_name	HDF-STRING	1	code	"1km Data Samples"
Per 1km Data Line in this MODIS Granule units	HDF-STRING	1	code	"1km Data Samples"
valid_range	HDF-uint8	2	code	0,1354
_FillValue	HDF-unit8	1	code	-1
Data Lines	HDF-uint16	1	code	Variable
Maximum Number of Data Samples Per Line	HDF-uint16	1	code	Variable
Nadir Data Resolution	HDF-STRING	1	code	"1km"

Figure F-1. MODIS Data Product File Definition Examples (Continued)